Video segmentation using motion and colour

Piek, M.C.

Award date:
2002

TECHNISCHE UNIVERSITEIT EINDHOVEN

Department of Mathematics and Computer Science

MASTER'S THESIS

Video segmentation using motion and colour

by

Matthijs Piek

Supervisor:	dr. ir. H. van de Wetering
Advisors:	dr. H. Zantema drs. W. Nuij dr. C. Varekamp (Philips Research) ir. R. Braspenning (Philips Research) dr. G. Lunter (Philips Research)

Eindhoven, August 2002

Abstract

For various applications, for example data compression, 3D reconstruction and medical imaging, there is a demand for an algorithm to divide video sequences into objects. For still images, several approaches exist based on colour , but these lack in both speed and quality of the segmentation. Most likely, colour information is an inherently inadequate cue for real world object segmentation. For video sequences, however, an extra cue is available, namely apparent motion. Apparent motion is how real life motion appears on a screen. When different objects in a scene have different motion, the motion cue alone is often enough to reliably distinguish objects from one another and the background. However, because of the lack of sufficient resolution of motion estimators, the resulting segmentation is not at pixel resolution. From prior research [2] it is concluded that motion estimation and segmentation is particularly effective in video sequences with much detail. However, colour segmentation is particularly effective in video sequences with little detail. This led us to believe that much can be gained from their combination.

Our first challenge was to devise a suitable block segmentation method based on motion vectors. We developed a method based on the well-known K-means segmentation method [6] which was adapted to use connectedness constraints and affine motion models. A decision was made to perform a block resolution motion segmentation first and to refine this afterwards with a pixel resolution colour segmentation method. This prevents the oversegmentation which colour segmentation methods normally produce. Furthermore, it restricts the computationally expensive colour segmentation to a subset of the image.

Our second challenge was to devise a method for refining this block resolution segmentation into a pixel resolution segmentation. Our idea is to perform pixel resolution segmentation only at the edges of clusters (also called segments). If in a pair of adjacent blocks, one block is from a different cluster than the other, there has to be an object edge in that pair of blocks. This property allows us to do the pixel resolution segmentation on pairs of blocks. We present a method which avoids problems with bifurcations and blocks which are in multiple pairs.

For the development of an algorithm it is most helpful to have an objective measure for the quality of an algorithm. This was our third challenge. Unfortunately, such exact objective measures generally only exist for problems with an equally exact specification. Because our problem does not have such an exact specification, we decided to define a ground truth output which we find desirable for a given input. We define our measure for the segmentation quality as being how different our segmentation is from the ground truth. This measure enables us to evaluate oversegmentation and undersegmentation seperately. Also, it allows us to evaluate which parts of a frame suffer from oversegmentation or undersegmentation.

The developed video segmentation method performs well in the segmentation of independent moving foreground objects from each other and the background. It combines the strong points of both colour and motion segmentation in the way we expected. One of the weak points is that the segmentation method suffers from undersegmentation when adjacent objects display the same motion. In sequences with detailled backgrounds the segmentation will have trouble finding the true edges. Apart from the results for video processing, we think that some of the techniques may be useful for other segmentation problems.

Contents

Preface

This document is my Master's thesis and as such the result of a graduation project to obtain the degree Master of Science in Computer Science from the Eindhoven University of Technology. The proposal of this project initiated from Philips Research in Eindhoven and the major part of the work has been performed there in the Video Processing and Visual Perception group.

This document is about the development of a method which divides video sequences into several parts. These parts represent the objects which are portrayed in the video sequence. Such a method is called a video segmentation method. As such, the intended audience are people with a technical background and an interest in video segmentation or its applications.

I would like to thank: Gerton Lunter, Chris Varekamp and Ralph Braspenning for their creative ideas and excellent supervision at Philips Research, Huub van de Wetering for his role as university supervisor and for keeping me on schedule, Geert Depovere and Jack van Wijk for allowing me to do my graduation project in their groups and finally Rimmert Wittebrood for his ideas and for several pictures in this report.

Chapter 1

Introduction

1.1 Subject of this report

This report is the result of research into a method for *video segmentation.* Video segmentation is, at least in the scope of this report, a way of dividing all video frames of a video sequence in several parts. Ideally, each of these parts is a real separate foreground object of the particular video sequence. The problem is described in chapter 3. The major part of this document deals with the specification, development and analysis of such a method. Besides this, we also introduce a method for the comparison of segmentations.

This chapter covers the context in which this research project was initiated. Section 1.1 briefly covers the subject of this report. In section 1.2, we describe the research groups at the Philips Research Laboratory Eindhoven which are involved and what they have to gain from this research. In section 1.3 we summarise possible applications for the research and, lastly, in section 1.4 elaborate on the two most important applications.

1.2 Research groups

The research is part of ongoing research in both the Digital Signal Processing and the Visual Processing and Visual Perception groups. Especially the project 3D Analysis of 2D Moving Video Sequences of the Digital Signal Processing group has much to gain from our research. This project focuses on developing robust techniques for the extraction of 3D information from 2D video sequences. As a means for this extraction various video segmentation techniques have been developed in this group [7]. The focus of these segmentation techniques has been on using colour as an identifying feature for segments. As a result, the group has developed an expertise in colour segmentation.

The Visual Processing and Visual Perception group performs research for the improvement of the perceived quality of video. As means to this end video processing algorithms and architectures are developed and perception studies are being done. In this group, much research has been conducted on motion estimation and compensation. As such, the group has developed a particular expertise in that field. Video segmentation is seen as a potential technique for increasing the quality of the video processing. Much more so than the research done in the Digital Signal Procesing group, the research in this group is intended for use in consumer electronics products. This means that their algorithms are more constrained by efficiency requirements, such as processing power and memory usage.

The idea for this research was initiated when researchers from both groups identified the potential mutual benefit from using each other's research results. From this insight came the idea to combine both expertise fields and as a consequence this Master's thesis project was formulated.

1.3 Possible applications

Sever applications are possible For video segmentation, each with their own specific requirements. We will list a few, together with some of their requirements:

Navigation: Moving devices may be fitted with cameras to identify goals and objects to be avoided. Using the segmentation, autonomously moving vehicles could be constructed. This application requires the algorithm to be *real-time*, but as the economic cost of autonomously moving vehicles still is quite high the algorithm has little other computational limitations. Also, because information is available about the control, and in effect the motion, of the vehicle, the motion estimation for this application is a more constrained problem.

Video compression: Modern compression standards allow objects to be encoded once for an entire video sequence. An algorithm to extract objects from video would be invaluable for such techniques. Compression in itself does not have any hard efficiency requirements. The preference for compression is quality over computational efficiency.

Video surveillance: The security applications are similar to the navigation applications. There is a need to identify separate objects and track certain objects. This would allow for autonomously zooming and panning of security cameras. This application requires the algorithm to be real-time as well as relatively low-cost. An alternative application could be to scan through previously recorded video sequences to quickly identify interesting parts of subsequences.

Biomedical: With modern medical scanning techniques vast amounts of data can be acquired. To help medical staff analyse the data a good video segmentation technique would be most helpful. Required for medical applications is that the technique is robust and accurate. Constraining the algorithm to medical applications would help constrain the problem and would probably make it easier to salve the problem.

Video processing: Video processing is an application in which video sequences are used as input to produce other video sequences. Usually these output video sequences are a visual enhancement of the previous video sequences. Various video processing applications, like for instance temporal and spatial interpolation, could benefit from a good video segmentation. If the video processing is intended for a consumer electronics device the algorithm has strict efficiency requirements like low memory usage and the requirement that it has to run in real-time.

Object recognition: For computer vision applications it is often useful to recognise certain objects. A good video segmentation would allow objects to be tracked over time and thus possibly allow better recognition of the object. This means the application needs the video segmentation to reliably track objects over time.

Scene interpretation: Another category of applications is the interpretation of scenes. An important goal in this application category is to extract 3D information from 2D video sequences. It is important that the segmentation is robust and that it facilitates tracking of objects.

1.4 Intended applications

Although we want our segmentation method to be as generic as possible, it is hardly feasible to develop a segmentation method for all possible applications. We did not limit us to the two applications described furtheron in the text, but they are part of the context in which the algorithms were developed and as such have influenced our design decisions.

1.4.1 Video processing

Lots of video processing applications benefit from the ability to identify which pixel belongs to which object and the ability to track those objects over time. An example of such an application is temporal interpolation. Temporal interpolation raises the amount of frames in a video sequence by interpolating frames between existing frames. To allow this to happen with some accuracy, it is very important to know the motion of the objects. Motion of objects can be more easily estimated when it is known which parts of the screen belong to a particular object. Furthermore by tracking objects more information can be gathered for the interpolation of the frames. The most important requirement for this application is that it can be implemented in a consumer electronics product and therefore has to be able to run real-time. This severely limits the complexity and memory usage of an algorithm. Most video processing applications benefit from an accurate segmentation of the independently moving foreground objects and a way to identify the background. Video processing applications prefer oversegmentation to undersegmentation. Oversegmentation means that there are more segments than objects in a give frame. See also 3.4.3.

1.4.2 3D reconstruction

The research effort into 3D reconstruction is an effort to extract 3D information from a 2D video sequence. This goal can be useful for many applications. At Philips Research most research is done for one application in particular. This application is 3D television. Currently, several 3D displays are available, but there is little 3D content available. By being able to extract 3D information from existing two 3D video sequences, the lack of content would cease to be a problem. Furthermore, if 3D reconstruction could be done real-time in a consumer electronics product, these productiones could use conventional media as 3D content. If, by 3D reconstruction, each foreground object and the background have distinct marginally accurate depth values, a 3D display would be marketable according to perception studies. However, from similar studies it seems that these depth values are not allowed to fluctuate too much over time and that it is very important that no parts of the background get the same depth value as the foreground objects. For a segmentation method to be useful for 3D reconstruction it has to be able to track objects over time to allow objects to have similar depth values at subsequent frames. Also, 3D reconstruction prefers oversegmentation to undersegmentation and the segmentation made must be able to assign segments to at least the independent foreground objects with the background accurately separated from the foreground.

Chapter 2

Notation and definitions

In this chapter we introduce some notations and definitions which we will commonly use throughout this report. Section 2.1 covers the basic dimensions for input and output data. Section 2.2 covers the input data itself. Our representation of segmentation is covered in section 2.3 and the syntax we use for our algorithm is described in section 2.4.

2.1 Video dimensions

Input video exists of discrete samples which each have an unique position in time and space. A *pixel* is the smallest available element of a video sequence. A *video sequence* is a sequence of consecutive *video frames*. We assume that all frames in a sequence have the same width and height in pixels. The width of a frame in pixels is denoted by X. The height of a frame in pixels is denoted by Y. Frames are received as input one by one. The number of frames received at some moment in time is N. The current frame being processed will always be referred to as n. N increases at fixed time intervals. For television applications these time intervals are typically 1/24, 1/25, 1/50 or 1/60 of a second. So the domain D for video sequences at some moment in time is:

$$D = \{0, ..., X-1\} \times \{0, ..., Y-1\} \times \{0, ..., N-1\} \text{ with } X, Y, N \in \mathbb{N} \tag{2.1}$$

We refer to a position either explicitly as a triple $(x, y, n) \in D$ or as a pixel $p \in D$ when there is no need to be so explicit. Furthermore, when referring to positions as a triple, any of the coordinates may be omitted when they are unambiguous. Some input data may be given per *block*. All blocks are equally sized. The width of a block in pixels is $\delta_{\mathrm{x}}^{\mathrm{b}}$ and the height in pixels is $\delta_{\mathrm{y}}^{\mathrm{b}}$. Note that the subscripts x and y do not denote any dependency on the location of a block, but denote the horizontal and vertical size of a block. For the size of a block holds:

$$X/\delta_{\mathrm{x}}^{\mathrm{b}} \in \mathbb{N} \quad \wedge \quad Y/\delta_{\mathrm{y}}^{\mathrm{b}} \in \mathbb{N} \tag{2.2}$$

$$\delta_{\mathrm{x}}^{\mathrm{b}} \in \mathbb{N} \quad \wedge \quad \delta_{\mathrm{y}}^{\mathrm{b}} \in \mathbb{N} \tag{2.3}$$

The top left block of frame n starts at position $(0, 0, n)$. A block may be identified by any pixel in it. We refer to a block either explicitly as related to a pixel p by p_{b} or as a block b if there is no need to be so specific. This means the following holds for every pair of pixels (x, y, n) and (x', y', n'):

$$(x, y, n)_{\mathrm{b}} = (x', y', n')_{\mathrm{b}} \equiv$$
$$\lfloor x/\delta^{\mathrm{b}}_{\mathrm{x}} \rfloor = \lfloor x'/\delta^{\mathrm{b}}_{\mathrm{x}} \rfloor \ \wedge \ \lfloor y/\delta^{\mathrm{b}}_{\mathrm{y}} \rfloor = \lfloor y'/\delta^{\mathrm{b}}_{\mathrm{y}} \rfloor \ \wedge \ n = n' \tag{2.4}$$

In our algorithms it is useful to have a notation for the set of all pixels and blocks in a particular frame. For this reason we introduce the following sets.

$$D^n = \{0, ..., X-1\} \times \{0, ..., Y-1\} \times \{n\} \qquad \text{with } 0 \leq n < N \tag{2.5}$$
$$D^n_{\mathrm{b}} = \{0, \delta^{\mathrm{b}}_{\mathrm{x}}, ..., X-\delta^{\mathrm{b}}_{\mathrm{x}}\} \times \{0, \delta^{\mathrm{b}}_{\mathrm{y}}, ..., Y-\delta^{\mathrm{b}}_{\mathrm{y}}\} \times \{n\} \qquad \text{with } 0 \leq n < N \tag{2.6}$$

2.2 Input data

Our input data takes two forms: *colour* and *displacement*.

2.2.1 Colour

Colour is received for every pixel. Colour is represented as a triplet of scalar colour values. Two colour spaces are regularly used in video processing. These both have their unique representation. The RGB colour space represents each colour as an unique combination of red (R), green (G) and blue (B). The YUV colour space represents each colour as a combination of luminance (Y) and chrominance (U and V). Note that there exists a simple transformation for conversion between these two colour spaces. We define the following domains:

$$K_{\mathrm{YUV}} = \{0, ..., 255\} \times \{-128, ..., 127\} \times \{-128, ..., 127\} \tag{2.7}$$
$$K_{\mathrm{RGB}} = \{0, ..., 255\} \times \{0, ..., 255\} \times \{0, ..., 255\} \tag{2.8}$$

The colour data for each frame is defined by one of the following functions depending on the colour space used:

$$C_{\mathrm{YUV}} : D \mapsto K_{\mathrm{YUV}} \tag{2.9}$$
$$C_{\mathrm{RGB}} : D \mapsto K_{\mathrm{RGB}} \tag{2.10}$$

These extra functions are introduced to facilitate easy access to the components of the colours:

$$C_{\mathrm{YUV}}(p) = (C_{\mathrm{Y}}(p), C_{\mathrm{U}}(p), C_{\mathrm{V}}(p)) \tag{2.11}$$
$$C_{\mathrm{RGB}}(p) = (C_{\mathrm{R}}(p), C_{\mathrm{G}}(p), C_{\mathrm{B}}(p)) \tag{2.12}$$

When any of the colour space indications are omitted, the choice of colour space is inconsequential.

2.2.2 Displacement

Displacement is received for every block. Informally put, displacement is how a part of the screen appears to move from one frame to the next. Formally put, displacement is a two-dimensional

vector which illustrates a semantic correspondence between pixels in subsequent frames. As a function it is defined as follows:

$$\vec{d}: D \mapsto \mathbb{Z} \times \mathbb{Z} \times \{-1\} \tag{2.13}$$

$$\vec{d}(p) = \begin{pmatrix} d_x(p) \\ d_y(p) \\ -1 \end{pmatrix} \tag{2.14}$$

The one element set above and the bottom row of the vector indicate that the displacement for a pixel only illustrates a correspondence between that pixel and some pixel in the previous frame. Because displacement is received per block, the following holds:

$$(\forall p, q : p_b = q_b : \vec{d}(p) = \vec{d}(q)) \tag{2.15}$$

Pixel $p = (x, y, n)$ is an image of the exact same part of reality as q if the following holds:

$$q = p + \vec{d}(p) \ \wedge \ q \in D \tag{2.16}$$

Note that the displacement received as input is not perfect and may not satisfy equation 2.16.

2.3 Segmentation

Segmentation is indicated by attaching a label to each pixel. This label is defined by the following function:

$$S : D \mapsto \mathbb{N} \tag{2.17}$$

Two pixels p and q are in the same segment (also called a cluster) if and only if $S(p) = S(q)$.

2.4 Syntax

In this text, we will present our methods as algorithms. For the syntax of our algorithms we use a syntax derived from Dijkstra's Guarded Command Language, which was first defined in [4]. We will briefly describe this program notation, which we expect to offer no problem to anyone familiar with ALGOL60, Pascal, or even C.

The Guarded Command Language has the following elementary statements:

- *skip* which does nothing.
- $x := E$ which evaluates expression E and then assigns its value to variable x.

Out of these statements, new statements can be constructed. This is done in the following manner, for existing statements S_0 and S_1:

- S_0; S_1 which first executes S_0 and then S_1.

- The following construct is called the *alternative construct*:

  if $B_0 \rightarrow S_0$
  $[]$ $B_1 \rightarrow S_1$
  fi

  This executes S_0 when B_0 holds or executes S_1 when B_1 holds. The B's are called the *guards* and the S's are called the *guarded statements*. The construct may consist of any number of guards and corresponding guarded statements.

- The following construct is called the *repetitive construct*:

  do $B_0 \rightarrow S_0$
  od

  This is a loop which keeps executing S_0 as long as B_0 holds. It terminates when B_0 no longer holds. Similar to the alternative construct this construct may consist of any number of guards and corresponding guarded statements.

- To this we added the following syntactical shortcut, in which B is a boolean function:

  foreach $B(a)$ **do**
      S_0
  od

  Although this construct is not part of the Guarded Command Language, it can be expressed in the Guarded Command Language as follows:

  $A := \{a \mid B(a)\}$;
  do $|A| \neq 0 \rightarrow$
      $a :=$ an element of A;
      S_0;
      $A := A \setminus \{a\}$
  od

  This assumes A is not used elsewhere in the program text.

Chapter 3

The problem

As shown in chapter 1, there is a demand for an algorithm to divide video sequences in the parts which, together, form the images. Ideally, it would be possible to divide these sequences in such a way that we are able to extract the real separate foreground objects which are imaged in the video sequence. Unfortunately, it is not easy to define what real separate objects are. For lack of a better definition we will introduce them as objects which appear to move independently. We hope to compensate this weak definition with the models presented in later sections. The segmentation of the image into these parts is made by attaching a label to the smallest parts of the video sequences, the pixels. This label indicates which objects have contributed to this pixel. An algorithm which produces such a labeling is called a *video segmentation* algorithm. We have to note that, due to transparency and reflection, more than one object can contribute to a single pixel. Taking this into account would be incredibly complex, so for simplicity's sake we assume that the video sequences contain no transparency or specular reflection, hence every pixel can be attributed to a single object. A major part of television sequences contain no or little transparency or specular reflection and for many applications their impact is negligible. The data flow for the problem is depicted in figure 3.1.

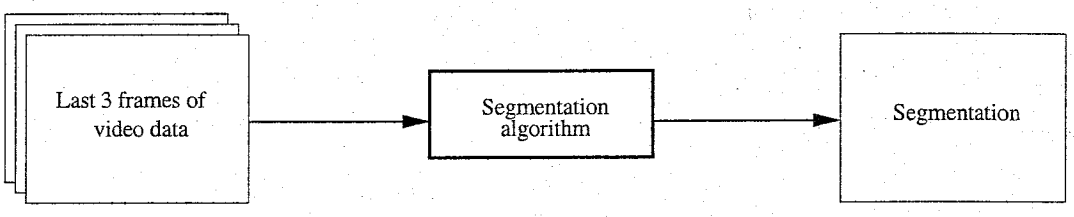

Figure 3.1: Blockdiagram for the problem of video segmentation. The arrows indicate the flow of data.

We want to partition our picture data in a number of *segments* in such a way that every pixel belongs to exactly one segment. We would like to have these segments correspond to the seperate natural objects as good as possible. In other words, we want every pixel p to have a corresponding label $S(p)$ in such a way that each pair of pixels p and q adheres to the following condition:

$$(S(p) = S(q)) \equiv (\text{pixel } p \text{ is part of an image of the same object as pixel } q) \tag{3.1}$$

This just leaves us with the task to define what an object is. Because, as far as we know, there is no useful definition for an object, we tried to characterise objects by formulating some assumptions. We assume the objects to adhere to some or all of the models presented in the next sections.

Furthermore, we want our algorithm to be able to be implemented to run in a consumer electronics

product. To allow this, we have to demand that an implementation in the not too distant future can run in *real-time* with a limited amount of memory. Real-time means that the algorithm, after an initial set-up time, has to be able to process a frame in the interval which exists between the starts of two subsequent frames. Furthermore it has to process the frames one by one in time-sequential order.

Apart from this, we would like the algorithm to be *time-stable*. By time-stable we mean that the segmentation at frame n should be similar to the segmentation at frame $n-1$. Time stability allows the application to track objects across frames by using *frame coherence*. Frame coherence expresses how much two consecutive frames relate to each other. Studies have shown that time stability is especially important for visual perception.

3.1 Models for motion

Several methods exist to extract motion from a video sequence. In order to present an interpretation of the motion vectors extracted by such a method, we present some models for motion. For this, we introduce the notion of *apparent motion*. Apparent motion is how real motion appears on the screen. So, apparent motion usually is the real three dimensional motion projected on a two dimensional plane. For instance, two objects can appear to have the same motion on the screen, while in fact one of the objects moves faster but at greater distance. Related to apparent motion is *displacement*, which we introduced in chapter 2. The concept is shown in figure 3.2. Similar to the apparent motion, displacement is three dimensional translation projected on a two dimensional plane.

3.1.1 Internal motion continuity

We will assume the displacement inside an object to be continuous. This is an assumption which holds for almost all objects. It fails for objects which have a discontinuity in distance to the screen along its visible surface. As visible in figure 3.2 displacement inside an object is continuous, when:

- The object displays translational movement in any direction (including motion towards or away from the camera).
- The object rotates around any axis.
- The camera displays translational movement in any direction.

In the sequel, we call this assumption the *internal motion continuity* assumption.

3.1.2 External motion discontinuity

To identify objects by motion within a frame, we assume that separate objects have an apparent motion that is distinguishable from the apparent motion of other objects. This assumption fails because not all objects display an apparent motion which is unlike the apparent motion of another object. However, in an attempt to simplify the problem, we choose to limit ourselves to this approximation. This allows us to separate the objects from each other by looking for motion discontinuities. In the sequel we refer to this assumption as the *external motion discontinuity* assumption.

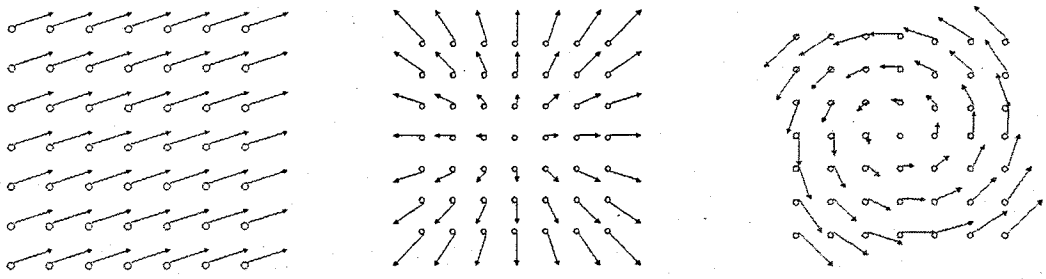

Figure 3.2: Displacement per pixel for an object with constant distance to the camera. The first image shows pure translational movement of the object and/or the camera. The second image shows an object moving away from the camera or a camera zooming out. The third image shows rotation of the camera or the object.

3.1.3 Affine motion models

A more specific model assumes every object to display apparent motion equal to an affine transformation. An affine transformation is a geometrical transformation which preserves lines and parallelism between lines. This means that if two lines in an object are parallel before the transformation they will be parallel afterwards. The general model allows *rotation*, *translation* and *disproportional scaling*. The equation, for *displacement* vector $\vec{d}(x,y)$ at position (x,y), is:

$$\vec{d}(x,y) = \begin{pmatrix} s_x & r_x \\ r_y & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{3.2}$$

In this equation t_x and t_y define the translation, s_x and s_y the disproportional scaling and s_x, s_y, r_x, r_y together define the rotation. Scaling is the motion in which an object appears to change size. Disproportional indicates that the ratio of width and height is not guaranteed to remain constant. Scaling is most often caused by either zooming of the camera or by objects moving closer to or further away from the camera. When the horizontal and vertical sampling density of an image are equal, zooming and any three dimensional translational movement causes an equal s_x and s_y. However, the horizontal and vertical sampling densities are hardly ever equal for television sequences, due to the use of various aspect ratios with fixed sampling frequency. But nonetheless these two kinds of movements cause the ratio s_x/s_y to be constant over the whole sequence. This ratio is equal to the ratio of the horizontal and vertical sampling densities. Unfortunately, this ratio is hardly ever known beforehand. We will use this model in three variations, based on the types of movement we assume to identify separate objects.

- The first is the two parameter affine model which describes translational apparent motion by assuming s_x, s_y, r_x and r_y are zero. Although this does not model all motion, this model may be an useful approximation, especially for smaller object or smaller parts of objects, because locally any motion is purely translational.
- The second is the four parameter affine model which describes all three-dimensional translational movement by assuming r_x and r_y are zero. This model describes a subset of motion caused by the internal motion continuity assumption. Motion covered by this model is most common in television sequences.
- The third is the six parameter affine model which also describes rotational movement. Rotational movement is less common in television sequences than translational movement. Mainly because it is hardly ever caused by camera motion.

3.1.4 Object inertia

From real world observation it seems that a reasonable and useful assumption to make is that objects have inertia. This implies that an object's motion at frame n is similar to its motion at $n-1$. This is a very reasonable assumption as many video sequences display real world objects and real world objects have mass and mass causes objects to have inertia. This assumption is referred to as the *object inertia assumption.*

3.2 Colour models

The colours of an object usually display some repeating patterns, gradients or other similarities. Unfortunately it is hard to define a measure which will quantify this for the colours of pixels inside an object. As an approximation we will introduce the next assumptions to indicate that the colours of pixels correlatee better with the colours of pixels inside the same objects than pixels outside. Similar to motion models we introduce:

internal colour continuity assumption: This assumes that there are no colour discontinuities inside an object. This is hardly ever the case. However, usually, there are just a very limited number of discontinuities inside an object, implying that using this assumption, would produce a slightly higher number of segments than objects.

external colour discontinuity assumption: This assumption states that every object is bordered by a colour discontinuity. In most video sequences this is the case.

constant internal colour assumption: Each pixel of an object adhering to this assumption has the same colour. This generally never happens, except in generated images like cartoons. As an approximation however it seems to be useful. Also, these objects are a subset of the objects adhering to the internal colour continuity assumption. Because of this, it also suffers from the same problems as that assumption.

3.3 Morphological models

We assume that separate objects can be distinguished from each other by some *morphological model.* Morphological models refer to the form of an object. In our case, we look at how the form of an object manifests itself after a projection on a two dimensional screen.

3.3.1 Connectedness

A common assumption to make is that objects are *connected.* This means that between every pair of pixels in the object, there is a path of pairwise adjacent pixels also completely contained in the object. There are two common variants of this assumption, *4-connectedness* and *8-connectedness.* In the *4-connectedness* variant every two consecutive pixels in a path must be either on the same horizontal or the same vertical line. In the *8-connectedness* variant two consecutive pixels in a path must be either on the same horizontal, vertical or diagonal line. We will call these assumptions the *4-connectedness assumption* and the *8-connectedness assumption.* To allow us to use 4- and 8-connectedness in our algorithms we introduce the following pixel- and block-*neighbourhoods*:

$$N_4(p) = \{q \mid (|q_x - p_x| + |q_y - p_y| \leq 1) \wedge (q_n = p_n) \wedge (q \in D)\} \quad (3.3)$$

$$N_8(p) = \{q \mid (|q_x - p_x| \leq 1 \wedge (|q_y - p_y| \leq 1 \wedge (p_n = q_n) \wedge (q \in D)\} \quad (3.4)$$

$$N_4^{\mathrm{b}}(b) = \{c \mid (\lfloor \frac{1}{\delta_{\mathrm{x}}^{\mathrm{b}}} |c_{\mathrm{x}} - b_{\mathrm{x}}| \rfloor + \lfloor \frac{1}{\delta_{\mathrm{y}}^{\mathrm{b}}} |c_{\mathrm{y}} - b_{\mathrm{y}}| \rfloor \leq 1) \wedge (b_{\mathrm{n}} = c_{\mathrm{n}}) \wedge (c \in D_{\mathrm{b}})\} \quad (3.5)$$

$$N_8^{\mathrm{b}}(b) = \{c \mid (|c_{\mathrm{x}} - b_{\mathrm{x}}| \leq \delta_{\mathrm{x}}^{\mathrm{b}}) \wedge (|c_{\mathrm{y}} - b_{\mathrm{y}}| \leq \delta_{\mathrm{y}}^{\mathrm{b}}) \wedge (b_{\mathrm{n}} = c_{\mathrm{n}}) \wedge (c \in D_{\mathrm{b}})\} \quad (3.6)$$

3.3.2 Occlusion

The connectedness assumptions are reasonable assumptions to make, however, they can be invalidated by *occlusion*. Occlusion occurs when one object moves in front of another in a video sequence or the camera moves in such a way that one object covers the view of another. Also, for the sake of generality, everything outside an image is considered to be an object too. This means that *clipping*, the phenomenon in which an object is only partly visible in the video frame due to the limited projection area, is a special case of occlusion. Occlusion makes it hard to track specific objects over time. There are two forms of occlusion, *partial occlusion* and *total occlusion*. Partial occlusion means that some part of the object is still visible and total occlusion means that the object is not visible at all. During partial occlusion, it is possible that two visible parts of the same object are not visually connected anymore.

3.3.3 Minimum size

To allow for a decent segmentation, we feel we also should propose a minimum size for objects. Without such an assumption it is nearly impossible to distinguish, for instance, noise from small objects. Because a great deal of the applications do not really need information about small objects this is also quite a reasonable assumption. Therefore we define the minimum size of an object to be one block.

3.4 Measuring segmentation quality

For the development of an algorithm it is most helpful to have an objective measure for its quality. Unfortunately, such exact objective measures generally only exist for problems with an equally exact specification. Our problem, however, does not have such an exact specification, but is specified in measures as yet only perceivable by humans. For this reason we decided not to try to find an objective measure, but instead to manually define an output which we find desirable for a given input. We call this output the *ground truth*. Our ground truth is shown in 3.3. We define our measure as an error. The error for the segmentation is how different our segmentation is from the ground truth. This means that when the error is 0, it produces a perfect segmentation. Although because of the manual creation of the *ground truth* this is not an objective measure, it allows us to objectively measure how close the produced segmentation is to the ground truth.

3.4.1 Test sequences and ground truth

Because of the great effort needed to produce the ground truth and the limited time that was available, we decided to test our measure with a ground truth test set of two frames, each originating from a separate sequence. The *mummy* sequence consists of a man on horseback moving to the upper right of the screen, while the camera pans to the right. In the foreground are several gunmen. The sequence contains partial and total occlusion and some camera pan and zoom. The *football* sequence is a typical sequence taken from a football match. The camera zooms in on a pair of footballers and the ball in the middle of the sequence. The sequence contains several potential problems like small objects, partial occlusion, detailed background and camera zoom.

The camera zoom is an especially big problem as the independent movement of the football is quite small relatively. The ground truth is shown in figure 3.3.

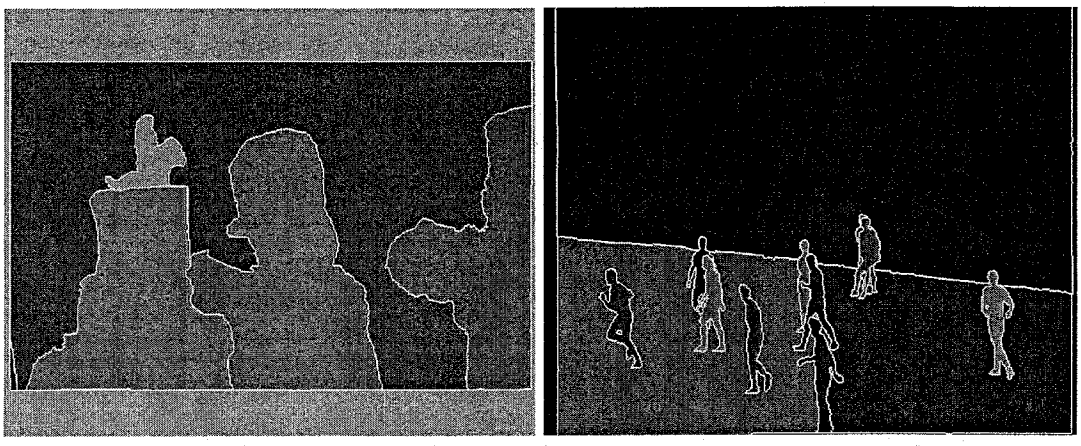

Figure 3.3: The ground truth for frame 13 of the *mummy* sequence and the *football* sequence. Edges between clusters are denoted as a white line.

3.4.2 Comparing a segmentation to the ground truth

To explain our measure, we first define a *border pixel* as a pixel which spatially neighbours a pixel of another cluster. An *edge map* is a two dimensional binary image in which these border pixels have the value true and all other pixels have the value false. Now for all border pixels from the ground truth edge map we calculate the distance to the closest border pixel from the produced segmentation. We define ϵ_u to be the average of all these distances. Vice versa, for all border pixels of the produced segmentation we will calculate the distance to the closest border pixels of the ground truth edge map. We define ϵ_o to be the average of all these distances. In [1] an efficient algorithm for the calculation of these distances is presented. Put into equations the following holds for the total segmentation error ϵ_s. The B sets denote the sets of border pixels for the ground truth and the segmentation. The GT function denotes the labels for the ground truth.

$$B_{\mathrm{GT}} = \{p|(p \in D^n) \wedge (\exists q : q \in N_4(p) : GT(p) \neq GT(q))\} \tag{3.7}$$

$$B_{\mathrm{S}} = \{p|(p \in D^n) \wedge (\exists q : q \in N_4(p) : S(p) \neq S(q))\} \tag{3.8}$$

$$\epsilon_o = \frac{1}{|B_{\mathrm{S}}|} \sum_{p \in B_{\mathrm{S}}} \min_{q \in B_{\mathrm{GT}}} \sqrt{(p_{\mathrm{x}} - q_{\mathrm{x}})^2 + (p_{\mathrm{y}} - q_{\mathrm{y}})^2} \tag{3.9}$$

$$\epsilon_u = \frac{1}{|B_{\mathrm{GT}}|} \sum_{p \in B_{\mathrm{GT}}} \min_{q \in B_{\mathrm{S}}} \sqrt{(p_{\mathrm{x}} - q_{\mathrm{x}})^2 + (p_{\mathrm{y}} - q_{\mathrm{y}})^2} \tag{3.10}$$

$$\epsilon_s = \epsilon_o + \epsilon_u \tag{3.11}$$

3.4.3 Interpreting the error measure

The two terms in the last equation, ϵ_o and ϵ_u, have a distinct interpretation. First, we define the notions of *oversegmentation* and *undersegmentation*. Oversegmentation occurs when in a certain area the produced segmentation contains more segments than the ground truth. Likewise, undersegmentation occurs when in a certain area the produced segmentation contains less segments than the ground truth. Because ϵ_o will be higher when the produced segmentation contains an

edge which the ground truth does not have, it is a measure for oversegmentation. Likewise, ϵ_u will be higher when the ground truth contains an edge which is not present in the produced segmentation. This means that it is a measure for undersegmentation. Undersegmentation and oversegmentation are illustrated in figure 3.4.

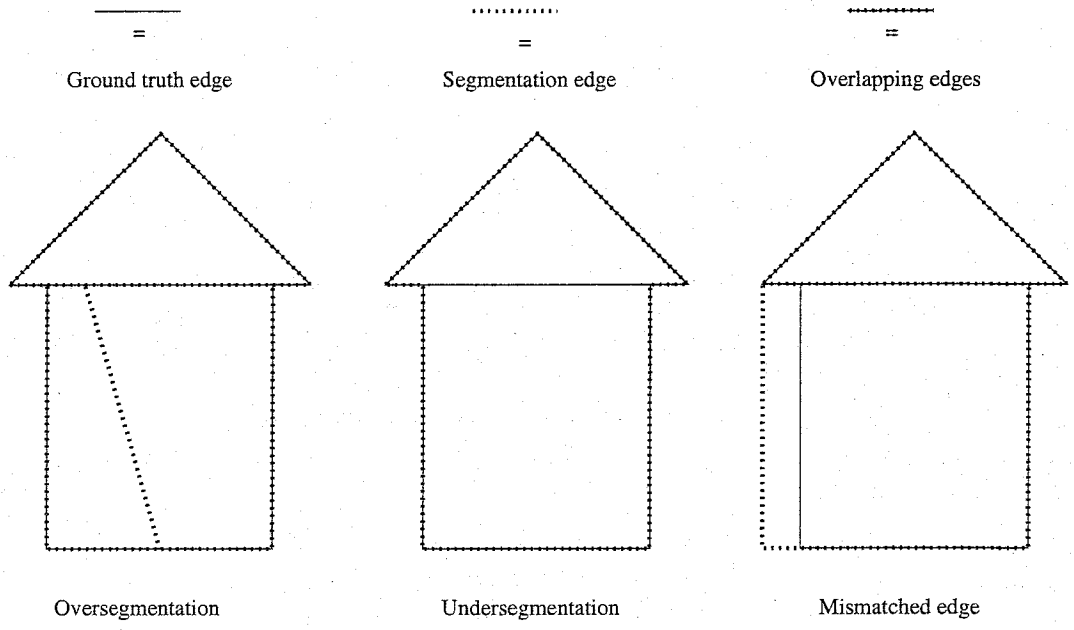

Figure 3.4: Oversegmentation and undersegmentation. Note that our measure yields an error of 0 for overlapping edges and increases ϵ_u for ground truth edges which do not overlap and increases ϵ_o for segmentation edges which do not overlap. This means oversegmentation increases ϵ_o, undersegmentation increases ϵ_u and mismatched edges increase both

These distinct interpretations allow us to evaluate the effects that changes in the algorithm have on undersegmentation and oversegmentation. This is especially useful because most applications prefer oversegmentation to undersegmentation. The measure enables us to quantify these aspects. Furthermore, it enables us to spatially plot the minimum distances, because border pixels in oversegmentation areas will produce a higher ϵ_o. Likewise, we can plot the undersegmentation errors. As a consequence the areas which cause problems with either over- or undersegmentation can be analysed.

Chapter 4

Approach

As seen in chapter 3 there are at least three cues available to create a segmentation: motion, colour and morphology . There has been extensive research into segmentation based on colour models. Most of these algorithm tend to produce a *oversegmentation*. An oversegmentation is a segmentation in which there are more segments than there are true objects visible in the image. The oversegmentation tends to be worse when there is a lot of detail in the picture, because most methods are based on the constant internal colour or internal colour continuity assumptions. This means that colour segmentation works best for images with little detail.

Most motion segmentation research has been conducted as a part of research into *motion estimation*. Motion estimation algorithms are algorithms which approximate the displacement of parts of the image between frames. Contrary to colour segmentation, motion estimation performs best on images with a lot of detail. Morphology is often used as support for motion or colour segmentation techniques. Because motion estimation performs best in pictures with high detail and colour segmentation performs best in pictures with low detail, it seems likely that much can be gained from combining the two techniques.

For motion estimation, there is a solution available which has been implemented in consumer electronics products. This method is called the *3D Recursive Search* (3DRS)[3]. When the motion vectors produced with this method are visualised, it is quite possible to extract the separate components visually. This fact inspired us to the idea that 3DRS might be helpful in producing a viable solution to video segmentation. Much more so than other motion estimation algorithms, 3DRS produces motion vectors which closely resemble the true displacement of the objects in the frame. Because of these aspects, we decided to base our algorithm on the 3DRS method. We use an extension of the motion estimation method, devised by Gerton Lunter, which uses three frames as input instead of the normal two. It is described in [5].

The 3DRS method divides an input frame in a regular grid of blocks sized 8×8 pixels. As a result, it returns a vector for each of these blocks. This vector should represent the main displacement vector for all pixels in the block. For many applications, most of these vectors are quite a good estimation of the true motion. When used for segmentation, however, 3DRS suffers from several disadvantages:

1. Because 3DRS is based on blocks, it is impossible to find the pixel resolution edges of objects based solely on the displacement vectors.

2. Due to its matching criteria, the accuracy of the produced vectors is especially low in homogenously coloured regions. In these areas "bleeding"-artifacts are usually visible. This means that parts of a homogenously coloured region will adopt vectors from another usually heterogenously coloured region. Colour homogeneity (and its opposite colour heterogeneity

or detail) is a measure for how much colour values vary in a specific region.

3. Although the estimated vectors found within a single object usually are similar, they are not exactly the same. So creating clusters from the vectors is not straight-forward.

Our job now is to find a method which will compensate these three disadvantages with the other two cues without raising the computational complexity too much. The observation that colour segmentation performs better in images with low detail and motion segmentation performs better in images with high detail is an important one. We want to achieve a good segmentation which does not suffer as much from all the cue's disadvantages. For this, we would like to propose the following skeleton for an algorithm.

As a first step in the algorithm, we will want to cluster all blocks with motion vectors which adhere to the same motion model. This will result in every single block in the frame being labeled with a cluster. As we now have created clusters from the vector field, we now have taken care of the third disadvantage. The design decision to start with the motion segmentation is an important one, as this allows us to prevent the oversegmentation which colour segmentation produces. Furthermore colour segmentation is relatively computationally expensive because of the need to inspect all pixels.

After this step, we try to find all blocks for which one of the neighbouring blocks has been labeled with a different cluster. Every block found this way is adjacent to a block which belongs to a different object. This means that the edge between these two objects, should be in this block or in the appropriate neighbouring block or both. Similar to border pixels, we will refer to these blocks as *border blocks*. These are the blocks in which we will have to do edge-detection, relieving us from the need of inspecting all blocks with a relatively computationally expensive pixel-based method. This multiple-step approach is illustrated in figure 4.1.

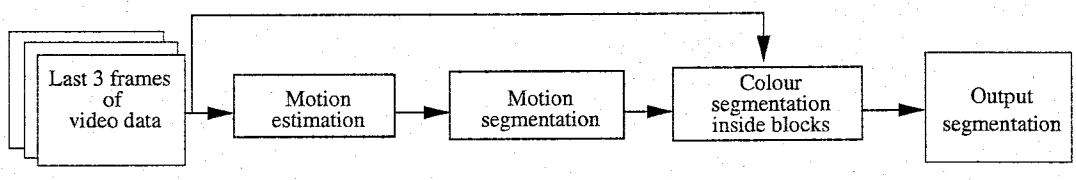

Figure 4.1: Blockdiagram for our approach. The arrows indicate the flow of data.

Before making a segmentation inside a block you can reason about the number of segments and where there should be segments inside a block. When a block is bordered by blocks from two clusters, for instance, there is a high probability that inside the block there will be two segments. Thus, a block is likely to have as many segments as it has bordering clusters. Also, if the left and bottom neighbours of a block are part of one cluster and the right and top neighbours are part of another cluster, it is more likely that the pixels on the left and bottom part of the block belong to the first cluster and that the pixels on the right and top part of the block belong to the second cluster. This can be generalised for any number of clusters. If the found segmentation agrees with these conditions, you can assign the appropriate intra-block segments to the appropriate clusters. If the segmentation does not adhere to these conditions, this can mean one of two things:

- The edge between a set of bordering clusters is not in this block, but in a neighbouring block. The segments in this block should be assigned to clusters based on information in neighbouring blocks.
- The motion vector was erroneous. This means there is no edge in this block or neighbouring blocks.

Choosing between these two alternatives can be done by checking if there is an edge in the neighbouring blocks. In the case of an erroneous motion vector, the clustering of the blocks has to be

changed. Based on some criteria, a choice has to be made about which cluster this block and its neighbouring blocks really belong to. One possible criterion is to test the motion model of the neighbouring blocks for the current block and label this block with the cluster corresponding to the best vector, but this is really similar to what already has been done in 3DRS. This takes care of both the first and second disadvantages.

Chapter 5

Block clustering

The first problem to be solved is the block clustering problem. The input for this problem are two (or three) last received frames of picture data and motion vectors for every block of the last frame. From these inputs, we want to label each block. This labelling should divide the frame into separate objects. We assume that each object adheres to some of the models presented in chapter 3. The process is higlighted in figure 5.1.

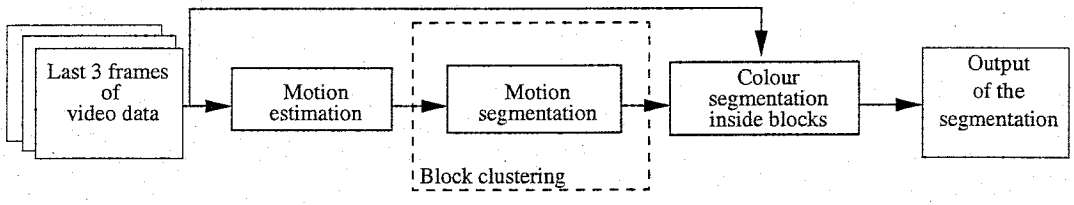

Figure 5.1: Blockdiagram highlighting the subject of this chapter. The arrows indicate the flow of data. The motion segmentation block has motion vectors as input and a labelling for every block as output.

Here we present some methods which cluster the data based solely on the motion vectors. Any pictures of segmented video frames show the edges between clusters as white lines. We will start with our first approach for tackling the problem and we develop the algorithm from there on by analysing the faults in the algorithm at each development step. In section 5.6 we will cover some ideas we have developed, but which do not contribute to the understanding of the final algorithm.

5.1 K-means

K-means is an algorithm devised for labeling an array of numbers with one of K unique labels. The algorithm calculates a mean for each label by averaging all numbers with that label. It minimises the summed error of all numbers to the means of their labels. It is not important of what type a label is, as long as each label is unique. K-means was first presented in [6] and has been used in numerous algorithms for video segmentation and other applications . The algorithm produces very nice results. When the algorithm is applied to a set of data which displays two clear peaks in its histogram, and K is 2, the sets will nicely encompass the peak and its direct neighbours. Furthermore, unlike other algorithms devised for similar purposes, it only requires one parameter, K, which is the number of sets (or clusters) to create. The algorithm is described as follows, in which the initial means I_k^0 have already been initialised to some value, one mean I_k^{-1} is unequal to the corresponding I_k^0, A is the set of values to be clustered and $i = 0$. We cover the initialization

more explicitly in section 5.2.

do $(\exists k : 0 \leq k <$K: $I_k^{i-1} \neq I_k^i) \rightarrow$
  foreach $a \in A$ **do**
    $label(a) := \arg\min_k(|I_k^i - a|)$;
  od;
  $i := i + 1$;
  foreach $0 \leq k <$K **do**
    $S_k := \{a \mid (label(a) = k) \wedge (a \in A)\}$;
    $I_k^i := 1/|S_k| \sum_{a \in S_k} a$;
  od;
od

If not all of elements are unique after the initialization, there is a possibility that one or more of the labels will be left unused. Furthermore, the initialization has a major influence on the number of iterations needed before the algorithm terminates. This implies that it may be worthwile to look for a way to make an initialization which approaches the final means as close as possible, without being overly complex.

5.2 Applying K-means for motion clustering

The good results achieved with this algorithm convinced us to try it for our purpose, assigning all blocks with the same label to the same cluster. However we needed to address some issues:

- The algorithm needs a definition for a mean for motion vectors. Using the normal definition of the mean, if N is the number of vectors used for the mean, the mean is the sum of all of the vectors scaled by $1/N$. This is suitable for our adaptation.
- We also need a measure for the difference between a motion vector d and a mean μ. For this, we have used the squared euclidean distance:

$$m(\vec{d}, \vec{\mu}) = (d_x - \mu_x)^2 + (d_y - \mu_y)^2 \tag{5.1}$$

  because the vectors describe spatial information. It is squared, because that is less expensive to calculate than the euclidean distance and yet does not affect the comparisons between the differences. Other options, like comparing angles in motion, may be viable for more specific applications. We will call this measure the *match*.
- Another necessity is a way to initialise the means. Because we are using video sequences, a logical way to initialise the means would be to copy the values of the means from the previous frame. This is likely to be a good guess for the means, because of the object inertia assumption. Furthermore, it introduces more temporal stability for the clustering. More temporal stability improves the chance that the same object will be in the same cluster in consecutive pictures. This approach uses the object inertia assumption.
- Also, we have to select the number of segments we want by choosing K. For now, we will choose K manually and later address the problem of selecting the right value for K based on the video sequences.
- This leaves the problem of the initialization of the means before the first frame, because then of course it is impossible to copy the means from the previous frame. As a solution

for first frame initialization we have chosen to divided the screen in K equally large regions. The means of the vectors in these regions are used as the means for initialization. This is illustrated in figure 5.2. Advantage for this approach is its low complexity and still reasonable efficiency. A disadvantage of this approach is that, regularly, some of the means will be the same, especially when large areas have similar displacement. If two means are the same, it is possible that there are two means which both are the best match for a block's vector. As the algorithm always assigns the lowest label, it can happen that the higher labels with the identical means will remain empty. We will address this problem at a later stage.

In the sequel we refer to this adaptation of the basic K-means algorithm as K-means.

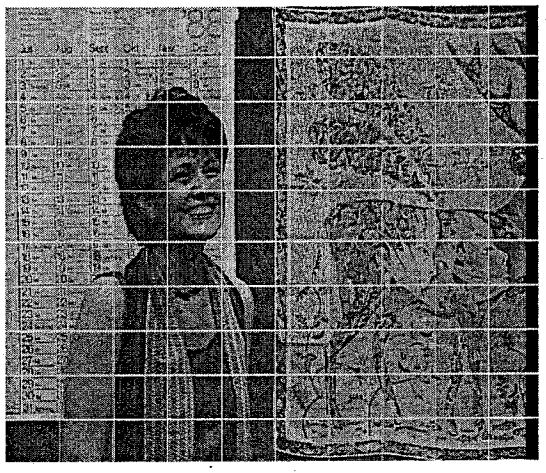

Figure 5.2: The initialization of the means before the first frame using $K = 100$. Each mean is initialised with the mean of the motion vectors in its region. Edges between regions are denoted with a white line.

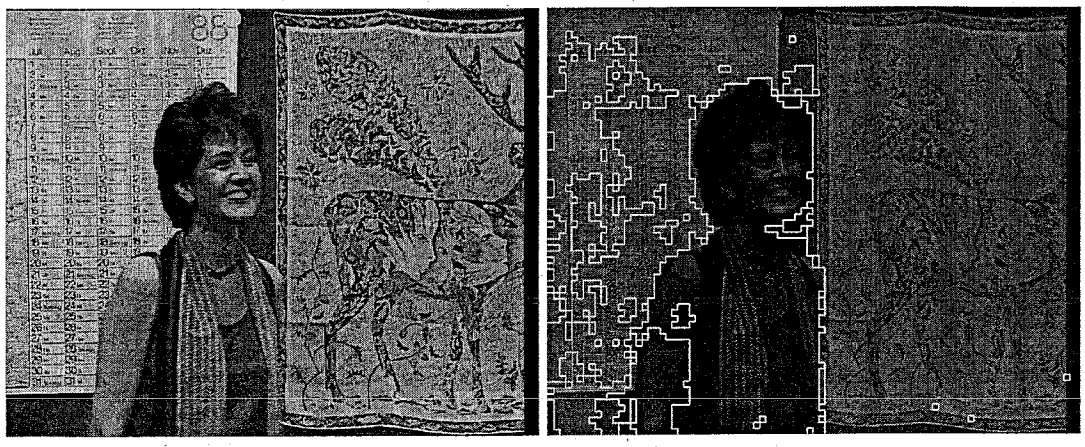

Figure 5.3: Vector K-means clustering of frame 19 of the *renata* sequence using K=9. Edges between clusters are denoted with a white line.

As you can see, this is a fairly straightforward way to adapt normal K-means for motion segmentation. K-means performs rather well. It easily separates objects adhering to the two parameter affine motion model assumption with clearly distinctive motion. One of the results is shown in figure 5.3. It is far from perfect however; it suffers from the following flaws:

- It tends to oversegment objects moving towards or from the camera and during camera zoom. Also rotating objects and objects which contain parts with different distances to the camera produce oversegmentation. Rotating objects are fairly uncommon in television video sequences.
- Objects displaying the same apparent motion for one or more frames tend to be assigned to the same cluster. In most cases, these objects are not even connected, so this may be partly remedied by using some connectedness criteria.
- Although noise in the vector field rarely is assigned to a cluster all by itself, it is frequently assigned to a cluster which does not represent the real object in that location. Noise usually only affects one block. This implies all block around the noise block will belong to a different cluster. If this is the case, a good connectedness criteria might help to remedy noise blocks from disturbing the clustering.
- Some clusters become empty because their initialised means are non-unique or because the object they described changed their motion to match the apparent motion of another cluster. Empty clusters do not have representative means and usually stay empty during next frames and iterations. We could try to prevent clusters becoming empty, but we have to note that, when an object has moved off the screen, empty clusters are part of the desired behaviour for the algorithm.

5.3 A new variation encouraging connectivity: K-regions

From the analysis of the K-means algorithm, it became obvious that encouraging connectedness might help remedy several of the faults experienced so far. See section 3.3.1 for definitions of connectedness. A solution for this is based on a novel idea from [8]. To encourage connectedness we have to extend K-means with a connectedness constraint. In this way we want to reduce the probability that clusters consist of non-connected blocks. We start the algorithm by initializing a certain connected region of blocks for every label. Then K-means is done, but with the one big exception that a block may only be assigned to a new cluster if one of its neighbours is in this new cluster In this way, it is less likely that the cluster will become unconnected. The algorithm is iterated until no block is assigned to a new cluster or a maximum number of iterations $i_{\max}$ is exceeded. This means the algorithm has reached a stable state. The algorithm, which we will call K-regions, is the following. In this algorithm we assume the regions to be initialised in a similar manner as the K-means algorithm, see section 5.2. The current frame is frame n. The variable i is initialised to 0 and $\vec{\mu}_k^{-1}$ is initialised in such a way that at least one $\vec{\mu}_k^0$ will not be equal to it. This algorithm uses the 4-connectedness assumption, but it can be easily adapted to use the 8-connectedness assumption.

Algorithm K-regions:
do $(\exists k : 0 \leq k <\mathrm{K}: \vec{\mu}_k^{i-1} \neq \vec{\mu}_k^i) \wedge (i < i_{\max}) \rightarrow$
  foreach $b \in D_{\mathrm{b}}^n$ **do**
    $C := \{k \mid (S(c) = k) \wedge c \in N_4^{\mathrm{b}}(b)\}$;
    $S(b) := \arg\min_{k \in C} m(\vec{\mu}_k^i, \vec{d}(b))$;
  od;
  $i := i + 1$;
  foreach $0 \leq k <\mathrm{K}$ **do**
    $S_k := \{b \mid (S(b) = k) \wedge (b \in D_{\mathrm{b}}^n)\}$;
    $\vec{\mu}_k^i := 1/|S_k| \sum_{b \in S_k} \vec{d}(b)$;
  od;

od

Note that the order in which the blocks are processed does influence the results. We processed the blocks from left to right and top to bottom. We can initialise the regions and means with the motion compensated region from the previous frame and the means from the previous frame. Motion compensating is done by using the vector field produced by 3DRS to find which block in the previous frame each block corresponds to. If the vector field is correct, it will allow us to find the previously assigned cluster for this part of an object except for parts which suffered from occlusion in the previous frame. This motion compensation is done by using the following initialization for each position in frame n:

$$S(p) = S(p + \vec{d}(p)) \tag{5.2}$$

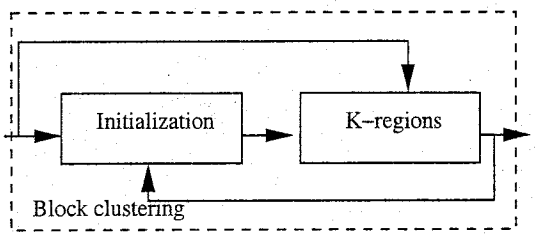

Figure 5.4: Blockdiagram highlighting the subject of this chapter. The arrows indicate the flow of data. This illustrates that the initialization depends on the previous segmentation.

The process is shown in figure 5.4. These are the results of the measure when comparing K-means with K-regions:

		K-means	K-regions
mummy	ϵ_o	0.74	0.70
	ϵ_u	0.64	0.66
	ϵ_s	1.38	1.35
football	ϵ_o	0.79	0.80
	ϵ_u	0.71	0.76
	ϵ_s	1.50	1.56
Total	ϵ_o	1.53	1.50
	ϵ_u	1.35	1.42
	ϵ_s	2.88	2.91

As you can see, this does not necessarily improve the results according to our measure. However, from visual inspection of several video sequences, K-regions clearly improves results. The bad results of the error measure can be explained. Because in the *football* sequence most of the clusters are in the bottom half of the frame, the added morphological constraint actually hinders the correct segmentation of the frame. Overall, K-regions seems to improve results though. A way to merge clusters in areas where they are not needed should remedy this problem. We cover such a method in section 5.5.

Because of the added motion compensated morphological information, this increases time stability over the K-means algorithm. When analysing the results achieved with K-regions, it seems, overall, to produce better results than the vector K-means algorithm. One of the results is shown in figure 5.5. Especially noise in the vector field is less of a problem and separate objects with similar motion will be assigned to distinct clusters more often, although objects with similar motion which are, or have been, in close proximity of each other sometimes get wrongly assigned to the same cluster. Unfortunately it has two distinct issues which need solving:

- Firstly, when a new object appears, it will not always get its own cluster or perhaps usurp another objects cluster. This is an especially big problem when an object appears within another object. This is so because the clustering is only updated at the edges of clusters. To solve this issue we need a criterion for introducing new clusters for the image.
- Secondly, when an object is totally occluded in a sequence, the cluster corresponding to that object will become empty. This is, of course, desired behaviour of the algorithm. However, when the object reappears, the cluster will still remain empty. Good criteria for introducing new clusters and removing old ones would probably also solve this problem.
- Thirdly, similar to the K-means algorithms, objects tend to break up in multiple clusters when they display motion towards or away from the camera or rotation movement.

The first two of these problems are adressed in section 5.5.

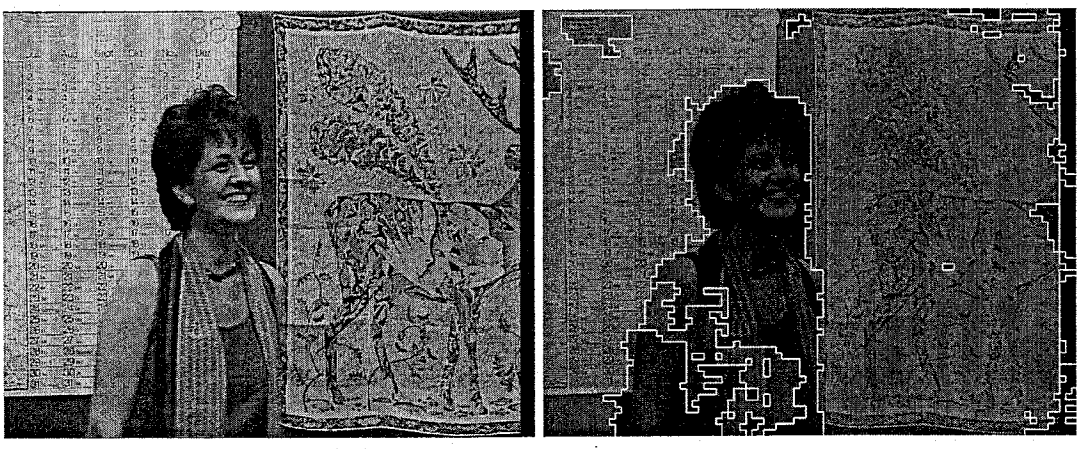

Figure 5.5: K-regions clustering of frame 19 of the *renata* sequence with K=9. Edges between clusters are denoted with a white line.

5.4 Using more motion models

One of the persistent problems experienced with all the algorithms concerns camera zoom and object motion toward or from the camera. When a sequence contains camera zoom, several objects will be divided in a high number of segments resulting in serious oversegmentation. Even worse, when the objects itself display relatively little motion while there is camera zoom, the segmentation will be really bad. The screen will just be divided in equally sized chunks in the worst case. As we have seen in section 3.1, there are motion models, especially the four parameter affine motion model, which capture this motion well.

To benefit from this, the algorithms need to be adjusted in such a way that it is possible to use it with an arbitrary motion model. This can be accomplished by two adjustments:

- Instead of the old criterion for relabeling blocks, we use the motion models to calculate the predicted displacement vectors at this block's location. A predicted displacement vector for model k at location (x, y) is the displacement vector some model will yield when x and y are used as the location. It will be denoted by $\vec{d}_{\mathrm{p}}^{k}(x, y)$. For the affine motion models this is (from section 3.1):

$$\vec{d}_{\mathrm{p}}^{k}(x,y) = \begin{pmatrix} s_{\mathrm{x}} & r_{\mathrm{x}} \\ r_{\mathrm{y}} & s_{\mathrm{x}} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_{\mathrm{x}} \\ t_{\mathrm{y}} \end{pmatrix} \tag{5.3}$$

When you've calculated the predicted vector, you can label the block with the label whose predicted vector produced the best match.

- Instead of calculating a mean like in the old algorithms, the model has to be updated to best fit the blocks which now have the appropriate label. There are several way to do this, dependent on the particular model. For the affine motion models described in section 3.1 the *least squares* method can be used.

When we apply these modifications to K-regions the algorithm looks like this:

Algorithm K-models:
do $(\exists k : 0 \leq k <\mathrm{K}: \vec{\mu}_k^{i-1} \neq \vec{\mu}_k^{i}) \wedge (i < i_{\max}) \rightarrow$
  foreach $b \in D_{\mathrm{b}}^{n}$ **do**
    $C := \{k \mid (S(c) = k) \wedge c \in N_4^{\mathrm{b}}(b)\}$;
    $S(b) := \arg\min_{k \in C} m(\vec{d}_{\mathrm{p}}^{k}(b), \vec{d}(b))$;
  od;
  $i := i + 1$;
  foreach $0 \leq k <\mathrm{K}$ **do**
    $S_k := \{b \mid (S(b) = k) \wedge (b \in D_{\mathrm{b}}^{n})\}$;
    "Update the model parameters with least squares";
  od;
od

When you have a model for your sample data, the least squares method allows you to determine parameters for this model in such a way that the square of the difference between model and data becomes minimal. This is done by first defining an appropriate error measure to the model with unknowns as parameters. A lower error measure should indicate a better match and an error of 0 indicates a perfect match. This should always be some kind of square error . When you sum these errors for all the sample data you get the total error, which is the term you want to minimise. This is simply done by differentiating the term with respect to each parameter, and requiring that the derivatives all equal 0. So we will create a set of equations by equaling these terms to 0 and solving the set for the parameters. This results in a closed formula for each parameter, depending only on the sample data. The values of these formulae will be the combination of parameters which yields the minimal total error.

If we apply this method to the affine motion models, you can acquire a simple expression for each parameter of the models, allowing the algorithm to update the models each iteration. This is the total error measure for cluster k's affine motion models, for sampled displacement vectors $\vec{d}$:

$$\epsilon_k = \sum_{b \in S_k} m(\vec{d}(b), \vec{d}_{\mathrm{p}}^{k}(b)) \tag{5.4}$$

For the 2 parameter model, which only models translational motion, assume s_{x},s_{y},r_{x} and r_{y} to equal zero. For the 4 parameter model, assume r_{x} and r_{y} to equal zero. Differentiating and solving the set of equations produces these useful equations for the 2 parameter model, in which $|S_k|$ is the number of samples, d_{x} and d_{y} are the horizontal and vertical component of the motion vectors and the summations are over all blocks modeled:

$$t_x = \frac{1}{|S_k|} \sum_{b \in S_k} d_x \tag{5.5}$$

$$t_y = \frac{1}{|S_k|} \sum_{b \in S_k} d_y \tag{5.6}$$

As you can see this is exactly equal to the mean of the motion vectors. So using our new algorithms with the 2 parameter model will produce exactly the same results as the old algorithms. The equations, with summations over all blocks modeled, for the 4 parameter model are:

$$t_x = \frac{(\sum x)(\sum d_x x) - (\sum d_x)(\sum x^2)}{(\sum x)^2 - |S_k|(\sum x^2)} \tag{5.7}$$

$$t_y = \frac{(\sum y)(\sum d_y y) - (\sum d_y)(\sum y^2)}{(\sum y)^2 - |S_k|(\sum y^2)} \tag{5.8}$$

$$s_x = \frac{(\sum d_x)(\sum x) - |S_k|(\sum d_x x)}{(\sum x)^2 - |S_k|(\sum x^2)} \tag{5.9}$$

$$s_y = \frac{(\sum d_y)(\sum y) - |S_k|(\sum d_y y)}{(\sum y)^2 - |S_k|(\sum y^2)} \tag{5.10}$$

The symbols are as above and x and y are the horizontal and vertical components of the coordinates of the centers of the blocks. The equations for the 6 parameter model can be found in a similar matter, but are too large to present here.

Using these equations we can perform both the K-means and K-regions algorithms with the three different motion models. Some of the results are presented in figure 5.6. The following table lists the results for the measure when we use motion models with the K-means algorithm.

		2 parameter K-means	4 parameter K-means	6 parameter K-means
mummy	ϵ_o	0.74	0.74	0.75
	ϵ_u	0.64	0.63	0.64
	ϵ_s	1.38	1.37	1.39
football	ϵ_o	0.79	0.82	0.82
	ϵ_u	0.71	0.65	0.66
	ϵ_s	1.50	1.47	1.48
Total	ϵ_o	1.53	1.56	1.57
	ϵ_u	1.35	1.28	1.30
	ϵ_s	2.88	2.84	2.87

The following table lists the results for the measure when we use the K-models algorithm above.

		2 parameter K-models	4 parameter K-models	6 parameter K-models
mummy	ϵ_o	0.70	0.71	0.70
	ϵ_u	0.66	0.65	0.64
	ϵ_s	1.35	1.36	1.34
football	ϵ_o	0.80	0.81	0.80
	ϵ_u	0.76	0.73	0.75
	ϵ_s	1.56	1.54	1.55
Total	ϵ_o	1.50	1.52	1.50
	ϵ_u	1.42	1.38	1.39
	ϵ_s	2.91	2.90	2.89

These numbers do not reflect it, but from experiments it seems that the 4 and 6 parameter models visually achieve extremely bad results when combined with the K-means algorithms. Many

clusters will be unconnected and it will be severely oversegmented. This can be explained by the fact that the 4 parameter model has more degrees of freedom to fit the sample data it is assigned to and thus matches the sample data over wrong parts of the screen. When combined with K-regions, the 4 parameter model achieves a real improvement over the results of 2 parameter model. Especially sequences with camera movement improve a lot, allowing K-regions to extract separate objects even if the zoom produces large motion vectors. Oversegmentation is greatly reduced because objects moving towards or from the camera will fit better into this model and do not tend to divide the object in separate clusters. Because of this, the clusters fit the seperate objects better and undersegmentation is reduced as well. Objects which have been assigned to the same cluster before, will now more often be assigned to seperate clusters. A disadvantage is that the algorithm still produces an oversegmentation with rotating objects, as we had expected. Overall however, the 4 parameter model improves the results over the old variations. As such, K-regions with the 4 parameter motion model is our preferred segmentation method. We refer to it as K-models.

Figure 5.6: K-models clustering of frame 21 of the *football* sequence with K=16. Edges between clusters are denoted with a white line. Top right picture is generated with a 2 parameter affine model, the bottom left picture with a 4 parameter affine model and the bottom right picture with a 6 parameter affine model. It is clearly visible that the 2 parameter affine model suffers from the camera zoom in this sequence.

5.5 Dynamically changing the number of clusters

Until now we have conveniently ignored that we do not know how many clusters an image will need to correctly represent all the objects. Although for some applications we may know the number of clusters beforehand, for most of our applications, for example video processing for television and 3D reconstruction, this information is unavailable. Also, the required number of clusters can be very dependent on the exact application.

Logically, when looking at the number of clusters, there are two distinct, problematic situations which can occur. Note that these can both occur in the same frame although at different locations:

Oversegmentation: Oversegmentation is what occurs when there are too many clusters compared to the number of objects. Usually this manifests itself in one object being subdivided in several different clusters. Causes of this are either that there are too many clusters available or that there are errors in the vector field. To counter this we need a criterion for *merging* clusters.

Undersegmentation: Undersegmentation occurs when there are fewer clusters than objects. Usually this manifests itself when two (or more) objects which are clearly different end up in the same cluster. This happens when a new object becomes visible from an occluded area, for example from an edge of the screen or from behind another object. Also, this happens when two objects which moved the same before, start behaving differently. For example when a basketballer throws a ball, the basketballer and the ball become separated. To counter this we need a criterion for *splitting* clusters.

Because the definition of an object is vague these notions are not exactly defined. This of course has its repercussions on the effectiveness of the merging and splitting of clusters. These notions have been defined more exact in relation to the ground truth in section 3.4.3. The ground truth itself though is of course highly subjective.

5.5.1 Merging clusters

As we want to counter oversegmentation, we most notably want to have larger clusters. However, we do not want the match of the motion models to worsen too much, because that would indicate that the motion of the two merged clusters is not similar and that they are probably not part of the same object. Note that, by merging clusters, the mean match of a cluster's motion model to the cluster's vectors will never improve. This happens because you are reducing the number of motion models by merging clusters. Reducing the number of motion models decreases the amount of freedom we have to match to the vectors. As a result of this, the mean error increases. This is why we thought it would be a good idea to only merge clusters when merging them would not increase the error too much over the old errors for the old clusters. Because of the connectedness assumptions, we only want to merge clusters which are adjacent to one another.

Luckily, because of the least squares error measure, it is not too complex to calculate what kind of an error a certain model produces in a certain area. This allows to easily calculate the error for the hypothetical merger of two clusters. If we check the increase of the error against a threshold t_m, we can avoid merging clusters which worsen the match too much. An adjacency graph can be constructed in a single scan over all blocks by storing which cluster neighbours which other clusters. When you combine these aspects, the merging algorithm looks like the following. Initially, $E^{-1} = \emptyset$ holds and E^0 is a set of the edges of the adjacency graph. The notation $l(c, d)$ denotes the label of edge (c, d). The algorithm Update-Label is introduced below the merging algorithm as a syntactical shortcut.

Algorithm Merge:

$i := 0$;
foreach $(c, d) \in E^i \rightarrow$
    Update-Label(c, d);
od;
do $E^i \neq E^{i-1}$
    $i := i + 1$;
    $E^i := E^{i-1}$;
    foreach $0 \leq k <$K**do**
        $E_k := \{(e, f) \mid ((e, f) \in E^i) \wedge (e = k)\}$;
        $d := \arg \min_{e|(k,e)\in E_k} l(k, e)$;
        $E_d := \{(e, f) \mid ((e, f) \in E^i) \wedge (e = d)\}$;
        if $(l(k, d) < t_m) \wedge ((d, k) = \arg \min_{(e,f)\in E_d} l(e, f)) \rightarrow$
            foreach $b \in S_d$**do**
                $S(b) = k$;
            od;
            foreach $(d, e) \in E_d$**do**
                $E^i := (E^i / \{(d, e)\}) \cup \{(k, e)\}$;
                Update-Label(k, e);
            od;
        fi;
    od;
od

Algorithm Update-Label(c,d):

$S_c := \{b \mid (S(b) = c) \wedge (b \in D^n_{\mathrm{b}})\}$;
$S_d := \{b \mid (S(b) = d) \wedge (b \in D^n_{\mathrm{b}})\}$;
$m := K$;
$S_m := S_c \cup S_d$;
Calculate a model for cluster m with blocks from S_m;
$\epsilon'_c := \sum_{b\in S_c} m(\vec{d}(b), \vec{d}^m_{\mathrm{p}})$;
$\epsilon'_d := \sum_{b\in S_d} m(\vec{d}(b), \vec{d}^m_{\mathrm{p}})$;
$l(c, d) := max(\frac{1}{|S_c|}(\epsilon'_c - \epsilon_c), \frac{1}{|S_d|}(\epsilon'_d - \epsilon_d))$;

Now we have a way to merge clusters, we have to find a suitable way to integrate this with our existing algorithms. There are three distinct positions in the algorithms at which we could do merging:

Before clustering: One of the possible options is to merge clusters before doing the real clustering. The advantage of this is that the number of clusters is reduced before clustering, making the clustering less vulnerable to noise and more capable of capturing the correct motion model. However, it has a big disadvantage as well. Firstly, at the start of a frame, the clusters do not have a good coherence to the current frame. This means that two clusters may well be incorrectly merged. In practice, this is such a huge disadvantage that merging before clustering is infeasible.

After clustering: As an alternative, it is a possibility to merge clusters after the clustering steps. This has none of the disadvantages if the merging is done before clustering. The only disadvantage is that it requires a segmentation which adheres to the external motion discontinuity assumption (see 3.1.2). Otherwise, it is possible that two objects will merge into the same cluster.

"During" clustering: A last alternative is similar to merging after the clustering steps with the

one big difference that you can start the clustering again on this segmentation. This has the advantage that the clusters will follow the general shape of objects better. A disadvantage is the raised complexity of adjusting the clustering again.

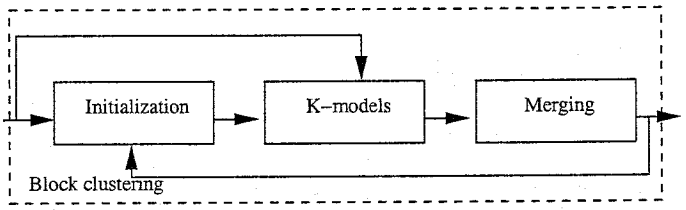

Figure 5.7: Blockdiagram of the block clustering process with merging. The arrows indicate the flow of data.

After evaluating these advantages and disadvantages we chose to do merging after clustering, resulting in a situation like in figure 5.7. Visually, the results are much better, because without merging lots of segmentations contain two adjacent clusters which both described the same object and were based on nearly similar motion models. This is shown in the top right part of figure 5.8. As expected the vague definition of objects caused the merging technique to be very hard to perfect. During experiments with this merging technique, it quickly became clear that it is very hard to determine a suitable threshold. A threshold which is set too high quickly results in an undersegmentation, while a threshold which is set too low can result in an oversegmentation. To complement the threshold, we have thought of a second measure for merging. This second measure should prevent clusters with a very low error from merging with clusters which have relative high error. This is measured by setting an extra threshold which is the old error multiplied with some factor. After some experiments, it seemed that we did not want to merge clusters if this caused errors to multiply by more than 3, although this number was also very hard to determine. Despite the problems with determining the threshold and the factor, merging seems to be a valuable, if not necessary, addition to this segmentation algorithm. Finding a really good method for merging might well require much more research, especially aimed at the improvement of the definition of an object. This is beyond the scope of this report.

5.5.2 Splitting clusters

To counter undersegmentation, we have to introduce more clusters. Because we had a relatively good way of merging clusters, a first, simple, idea was to introduce new clusters in a very simple way before clustering and have the merging method deal with all those clusters which had been created inappropriately after the clustering was finished. What we decided to do was to split all clusters in four parts, a top left, a top right, bottom left and bottom right part. This quickly introduces new clusters before clustering, allowing the new clusters to adopt to apparently new objects which may have appeared by utilizing the K-regions algorithm. Unfortunately, this did not seem to work, because of several reasons:

1. If a small new object appears in a large cluster, the newly created cluster will often be too large to adopt the new movement. Thus, the problem was that the newly created cluster was too large for the amount of new movement.
2. Our merging method did not seem to be good enough to always merge the clusters which were inappropriately split up, leaving an inappropriately high number of clusters.
3. The method also picks up smaller new changes in the vector field, which do not represent a new object at all. In other words, new clusters were created at the wrong location.

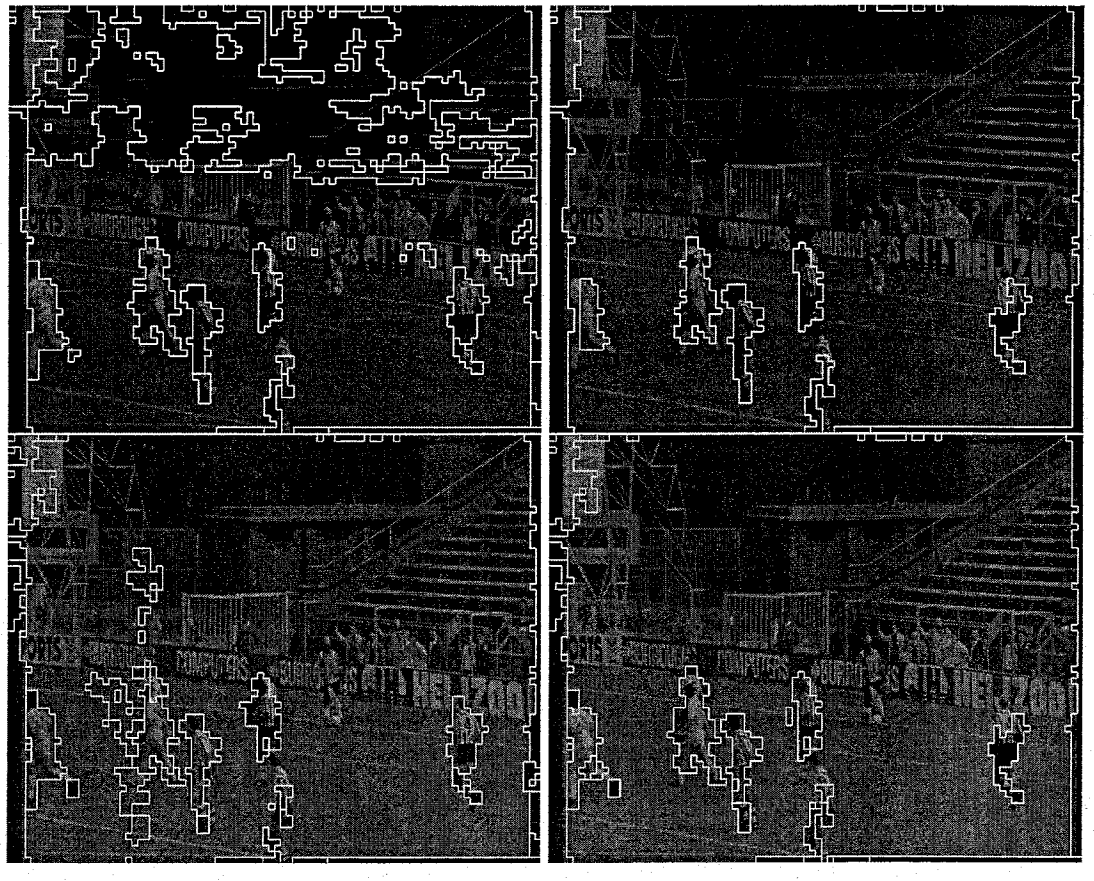

Figure 5.8: *Football* sequence frame 21: All pictures are generated with the K-models algorithm with $K = 16$ and the four parameter affine model. The top left picture is without merging or splitting, top right with the Merge algorithm, bottom left splits clusters around the block with the biggest error in each cluster, bottom right splits clusters at blocks which exceed a given threshold.

As this method clearly is not adequate, we tried to find a way to introduce new clusters while avoiding these three problems. To avoid these, we need a method which does not split too much when not appropriate, and forms a not too large cluster around an area where it is likely that a new object will be. When two (or more) objects share a cluster, it is likely that the blocks in which the error to the model is largest are part of another object than the motion model tries to represent. This idea led us to compare, for every cluster, the block with the biggest error to some threshold. If that threshold is exceeded a new cluster is initiated there. Because most of our motion models are not so representative when initialised with few blocks, we chose to have the clusters initialised with a larger three by three area around the block with the biggest error. Also, because we need the modeled information to be accurate, we split after the clustering step and do a second clustering step afterwards, to adapt the newly created clusters to the right shapes. This process is shown in figure 5.9. This method performed remarkably better than the previous, simple, algorithm. One of the results is shown in the bottom left part of figure 5.8. There are some drawbacks however. Firstly, because there is a threshold involved, the method is highly dependent of the choice of the threshold, just like the merging method discussed above. Secondly, it can only handle one new object per cluster per frame. Because, among others, the cluster representing the background often "creates" more than one new object per frame, this is an unacceptable disadvantage.

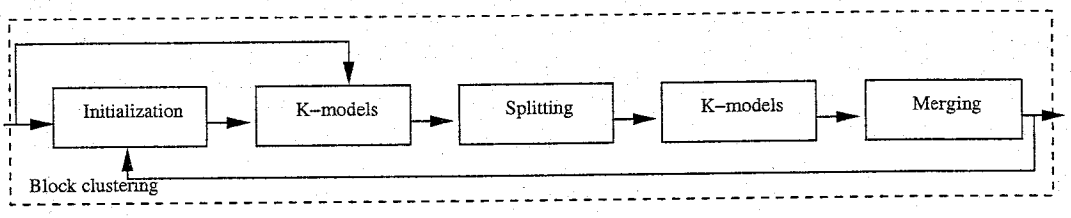

Figure 5.9: Blockdiagram of the block clustering process with merging and splitting. The arrows indicate the flow of data.

To counter the second disadvantage, we need to allow the creation of more than one object per frame. As we were already filtering with an error threshold, we decided that we should create objects for all blocks in the image which exceeded the threshold instead of the ones which contained the maximum error for the object as well. Also, using a three by three area around such a block seemed not the best way, so we decided to form a new cluster for each area of adjacent blocks with errors above the threshold. After some preliminary experiments, we noticed that the clusters created in such a way were a little too small, so we decided that some extra blocks should be added to the new clusters. These extra blocks are all blocks with errors exceeding half the threshold and which are immediately neighbouring one of the new clusters. If two new clusters get connected this way, they will be merged into one cluster. Results are shown in the bottom right part of figure 5.8. After experimenting with this splitting method, we concluded that, while it is hard to accurately pick the correct threshold, the method performs really well. Nearly all new discontinuities in the vector field will be assigned to a new cluster. Unfortunately errors in the vector field generate discontinuities as well. This means that most of the larger errors in the vector field will get new clusters assigned to them. This is something we can not solve by using the apparent motion cue alone. The following lists the various results for merging and splitting. Note that the first column is with merging only.

		Error increase merging	with maximum error splitting	with error threshold splitting
mummy	ϵ_o	0.70	0.72	0.72
	ϵ_u	0.66	0.67	0.64
	ϵ_s	1.36	1.39	1.36
football	ϵ_o	0.76	0.79	0.76
	ϵ_u	0.79	0.75	0.78
	ϵ_s	1.55	1.53	1.54
Total	ϵ_o	1.46	1.51	1.48
	ϵ_u	1.45	1.42	1.42
	ϵ_s	2.91	2.92	2.90

5.6 Sidetracks

This section contains the ideas which did not prove succesful for our segmentation, but may be worthwhile ideas for further research. The ideas presented here have failed in some way for our application.

5.6.1 Reducing the number of empty clusters in K-means

In an attempt to improve the quality of the K-means clustering (see section 5.2), we tried to enhance K-means. One of its most persistent problems is that clusters will become empty over time. To solve this, we allow clusters to be reinitialised. A way to reinitialise an empty cluster is by assigning it to the block which has the worst match with its current cluster. So for every block we have to calculate the difference between its displacement vector and its cluster's mean displacement vector. The block with the biggest difference is the block wich we're looking for. This implies that this is the most probable block to be part of another object than its cluster represents. If this block had a bad fit, because its displacement vector was affected by noise, the cluster will probably be empty again in the next frame. Because we will now regularly reinitialise clusters, clusters will not stay empty indefinitely. Also, because the blocks with the biggest errors are used for the reinitalization, clusters which contain two objects will be split in two more often.

Its disadvantage is that it decreases the time stability. The main cause of this is the reinitialisation of the clusters. After reinitialization there is no guarantee that the new blocks labeled with this cluster are in some way semantically linked to the blocks which were labeled with this cluster before. Furthermore, some times the block with the worst match really is part of the object its cluster represents. This is often the case when the object is moving towards or from the camera or during camera zoom. Visually, the method performs better than the basic K-means method introduced above as more of the initial segments will remain in use during the total sequence. A more generic approach is covered in section 5.5. This helps prevent *undersegmentation*. Undersegmentation is explained in section 3.4.3. This enhancement became obsolete when we introduced enhancements from section 5.5 in our algorithms.

5.6.2 Using more temporal information

When analysing the K-means algorithm (see section 5.2) we came to the conclusion that much might be gained by using information from previous frames. By using this information we wanted to try to improve the time stability of the algorithm. To improve time stability we have to try to increase the probability that an object will stay in the same cluster as it was in before. At the same time, though, we should allow an object to move into another cluster in case that the last clustering was incorrect. To allow this, we first have to know in what cluster this block was

before. Because of the motion, this is not as straightforward as selecting the cluster for the block at the same position in an earlier frame. One solution is to motion compensate the clustering, by using the motion compensation explained in section 5.3.

It is then possible to track a block over time, allowing the algorithm to keep data for a block across frames. This allows us to record the closest match the block b has ever had with a cluster by storing the difference with that mean at that time in a variable m_b. Now, to increase time stability, we will only allow this block to be assigned to another cluster, if the difference to that cluster's mean is smaller than m_b. This way a block will always stay in the cluster with which it has matched best. Unfortunately, although the idea sounds nice, it has two disadvantages. Firstly, the motion compensation tends to magnify structural errors in the vector field, because errors in the vector field mean a block will be attached to the wrong block in the previous frame and the corresponding wrong difference with its cluster. Secondly, new objects, which do not have any temporal information, will not be assigned to the right cluster as fast as before. The clear advantage of this method, though, is, that some, more temporary, errors in the vector field will not have any effect on the clustering. In some cases, this actually improves the vector information over the output of 3DRS.

In an effort to counter the second disadvantage, a method is introduced which reduces the effects of the best match over time. Every frame a certain positive term, called the *decay term*, is added to m_b for every block b. This means that, every frame, m_b will get worse and, thus, there will be a better chance that there will be a better match for block b in the next frame. In this way, blocks with older good matches are more likely to be assigned to a new cluster. Unfortunately, this affects the advantages of the method as much as the disadvantages, so it is very dependent on the exact properties of the used video sequence. As a high decay term is especially important in the first few frames of the sequence, we tried starting a sequence with a high decay term and gradually reducing this in subsequent frames. This did not work as well as expected, although it helped offset the earlier disadvantages of decay terms. However, it introduces an extra parameter for the algorithm, which is very important to the performance of the decay term. It turned out to be very hard to choose this parameter for a certain sequence. Because of this we decided not to use this. Using more temporal information might be worthwhile though and may be an interesting topic for further research.

5.7 Conclusion

The most important thing we can conclude from the experience we have with the methods above, is that motion indeed is a good cue for segmenting objects. It hardly suffers from the problems the colour cue has with oversegmentation, while also keeping undersegmentation to a minimum. Apparent motion is regularly less time stable than other cues though. While the colour of an object will only change marginally, the motion of an object typically changes quite much over small time periods. However, by doing motion compensation, it is feasible to track objects over time, which results in a better time stability than by using colour alone. This, however, is highly dependent of the accuracy of the motion estimator used.

Also, by further constraining clustering morphologically, better results are obtained. One of the advantages is that using morphological constraints allowed us to use more complicated models for the motion of objects. This conclusion might be a worthwhile conlusion not only for motion segmentation but also for other segmentation algorithms where morphological constraints may be valid.

Despite these positive findings there are some problems we have encountered:

1. If two (or more) objects display the same apparent motion, there is no way to avoid undersegmentation. Note that if at some time, they display different motion, some temporal

criterion may help to distinguish the objects. However, it may be very hard to find an appropriate criterion. This is an inherent problem of using motion as a cue.

2. The state of the art of motion estimation is not completely adequate for segmentation. Errors in the vector field cause problems with clusters containing lots of detail stealing blocks from other clusters with less detail. These errors also cause problems with new clusters appearing, while there is no reason for them to appear.

3. Furthermore, current motion estimation methods only return a vector per block, while a vector per pixel would be necessary for a pixel resolution video segmentation. Note that the methods we presented in this chapter will work with blocks of any size, including blocks sized one pixel. However, the current motion estimators force us to do a clustering on a block resolution.

4. Also, the accuracy of the motion vectors is not high enough, to reliably detect or distinguish small motion. This is especially a problem with objects moving towards the camera, because these will seem to differ in a relatively small way to the surrounding motion.

5. The block clustering method suffers from low temporal stability when objects reappear after total occlusion. This problem exists because information is stored only from one previous frame. Solving this problem would require the algorithm to store information on occluded objects and a technique to recognise objects from this information. The alternative we use now is to consider these objects as being new previously unseen objects.

6. The clusters representing objects with relatively small extremities, such as people at a distance, will often not encompass these extremities. This is probably because these extremities only encompass a relatively small part of a block, prohibiting 3DRS of finding the right vector for that block.

7. Merging and splitting produces, depending on the thresholds, under- or oversegmentation. As stated before this is most probably caused by the inexact definition of what an object is.

8. Large lowly detailled areas tend to be oversegmented, because the motion estimator returns incorrect vectors for these areas. This is caused because the motion estimators are based on matching and blocks with low detail do not offer enough information to find a reliable match in the previous frame.

Chapter 6

Intra-block clustering

We are now capable of producing a block clustering with the properties described in 5.7. This leaves us the challenge of creating a clustering at pixel resolution from this clustering. As presented in chapter 4, we will try to devise an algorithm which refines the edges of the clusters we found in the block clustering step. How intra-block clustering fits into the total scheme is shown in figure 6.1.

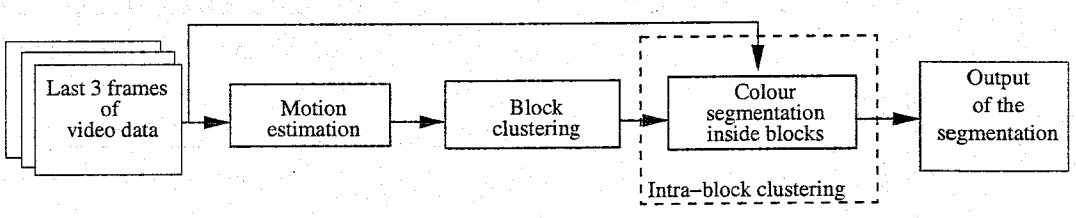

Figure 6.1: Blockdiagram highlighting the subject of this chapter. The arrows indicate the flow of data. The colour segmentation block has a block clustering and video data as input and a labelling for every pixel as output.

6.1 Edges

Our idea is to perform pixel resolution segmentation only at the edges of clusters. For this, we define *border blocks* as being those blocks which neighbour a block from another cluster. Similarly, we define *border pixels* as being those pixels which neighbour a pixel from another cluster. So, after block clustering, all border pixels will be inside border blocks. Also, in every pair of adjacent border blocks from different clusters, will be an object edge as long as the block clustering is correct.

This fact inspired us to do the pixel resolution segmentation on pairs of blocks. This should allow us to find all edges. In figure 6.2 a correct block clustering and the corresponding object edges are displayed. Of course, some blocks are in more than one pair of blocks. In this case we find all pixel resolution segmentations in some order and use the first segmentation as the initialization for the second, which is again used as the initialization for the third. Likewise, of course, for the third. This is illustrated in figure 6.3.

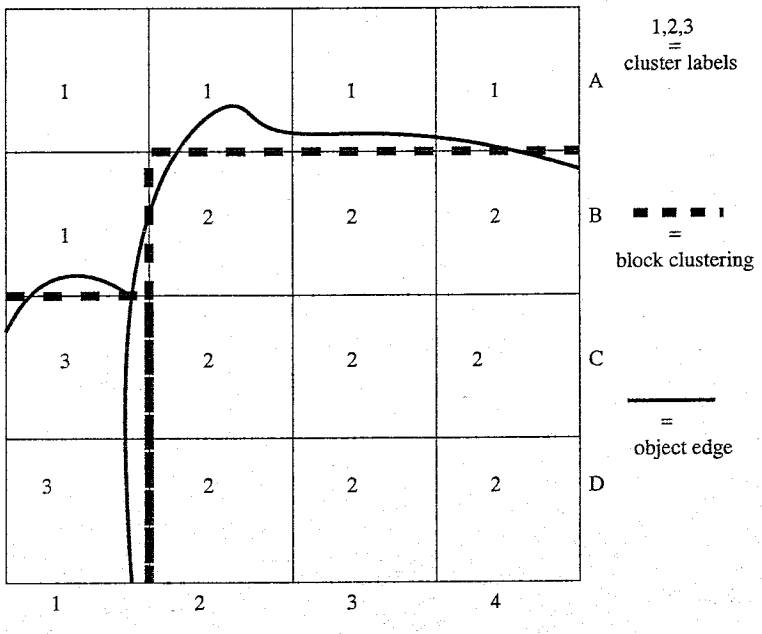

Figure 6.2: Correct block clustering and the real object edge

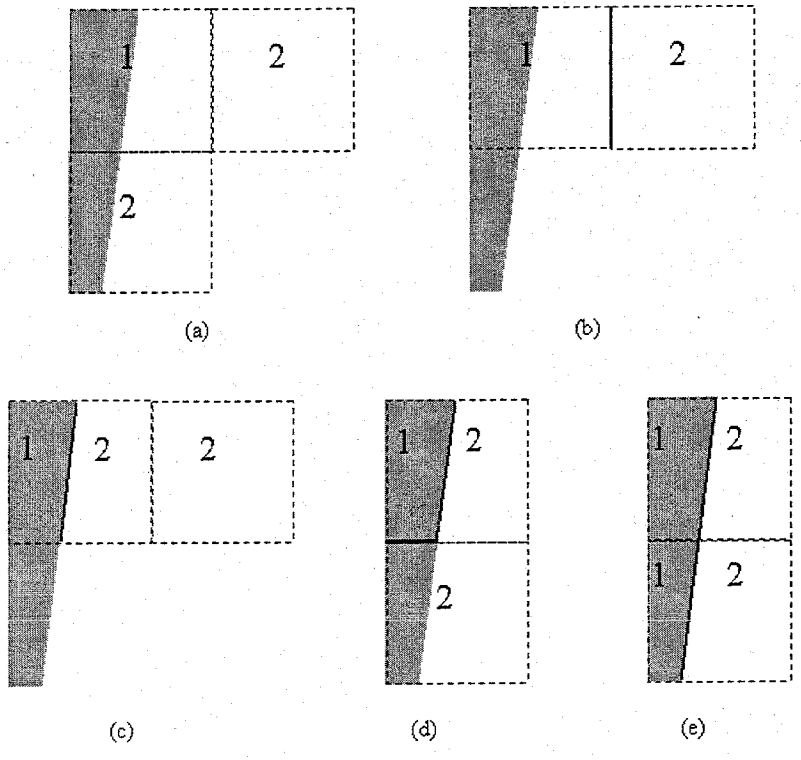

Figure 6.3: Initialization of pixel resolution segmentation involving different pairs of blocks. (a) Initial situation. (b) Initialization of the first pair. The black line represents the cluster edge. (c) Result of clustering. (d) Initialization of the second pair for this block. (e) Result of second clustering.

6.2 Pixel-resolution segmentation

Now we will introduce a method for doing the segmentation at pixel resolution. An advantage of already having a block clustering is that we know beforehand how many clusters the pair of blocks should contain. This allows us to use algorithms like K-means and K-regions without worrying about a heuristic to determine the value for K. In line with our positive experience with the K-regions algorithm for block clustering we chose to use the K-regions algorithm.

To allow us to use this algorithm we need some measure for colour, similar to the mean and distance for motion vectors. We would like to propose the following measures. The mean measures are just all components of the colours added and divided by the number of colours added. This works for YUV colour spaces as well as for RGB colour spaces. Testing indicates that it does not matter which colour space you use. In our case, it was YUV because that was most efficiently and easily implemented. For the distance, we would like to propose something like the squared euclidean distance:

$$m_{\mathrm{RGB}}(c,d) = (c_{\mathrm{R}}-d_{\mathrm{R}})^2+(c_{\mathrm{G}}-d_{\mathrm{G}})^2+(c_{\mathrm{B}}-d_{\mathrm{B}})^2, \quad \text{with } c,d \in K_{\mathrm{RGB}} \tag{6.1}$$

$$m_{\mathrm{YUV}}(c,d) = (c_{\mathrm{Y}}-d_{\mathrm{Y}})^2+(c_{\mathrm{U}}-d_{\mathrm{U}})^2+(c_{\mathrm{V}}-d_{\mathrm{V}})^2, \quad \text{with } c,d \in K_{\mathrm{YUV}} \tag{6.2}$$

Adapting the algorithm to use pixels instead of blocks and mean colour instead of mean motion the algorithm, executed for each pair b and c of adjacent border blocks of different clusters, becomes the following algorithm. In this algorithm we assume the regions to be initialised with the results of the block clustering, see chapter 5 or the previous pixel resolution clustering for any of the blocks, if available. The current frame is frame n. The variable i is initialised to 0 and μ_k^{-1} is initialised in such a way that at least one μ_k^0 will not be equal to it. This algorithm uses the 4-connectedness assumption and the YUV colour space, but it can be easily adapted to use the alternatives. We use component-wise summation and division for the summation and division of colours.

Algorithm K-colours:
do $(\exists k : 0 \leq k <\mathrm{K}: \mu_k^{i-1} \neq \mu_k^i) \wedge (i < i_{\max}) \rightarrow$
  foreach $0 \leq k <\mathrm{K}$ **do**
    $S_k := \{p \mid (S(p) = k) \wedge (p \in D^n)\}$;
    $\mu_k^i := 1/|S_k| \sum_{p \in S_k} C_{\mathrm{YUV}}(p)$;
  od;
  foreach $p \in \{q \mid (q_{\mathrm{b}} = b) \vee (q_{\mathrm{b}} = c)\}$ **do**
    $C := \{k \mid (S(c) = k) \wedge c \in N_4(p)\}$;
    $S(p) := \arg\min_{k \in C} m_{\mathrm{YUV}}(\mu_k^i, C_{\mathrm{YUV}}(p))$;
  od;
  $i := i + 1$;
od

Results on the renata sequence can be seen in figure 6.4. Results from our error measure are visible in the next table. As you can see, the results improve significantly with intra-block clustering.

Figure 6.4: Frame 10 of the *renata sequence*. The left picture is the block clustering. The right picture is the clustering resulting from the pair-wise intra-block clustering, using the K-colours algorithm. Lines which are darker indicate that the difference between the means during pixel resolution clustering was low. Note how lines are only darker where the pixel resolution segmentation found an incorrect edge.

		Input block clustering	Intra-block clustering
mummy	ϵ_o	0.72	0.60
	ϵ_u	0.64	0.26
	ϵ_s	1.36	0.86
football	ϵ_o	0.76	0.63
	ϵ_u	0.78	0.53
	ϵ_s	1.54	1.16
Total	ϵ_o	1.48	1.23
	ϵ_u	1.42	0.78
	ϵ_s	2.90	2.02

6.3 Compensating for incorrect block clustering

When analysing figure 6.4, you might notice that in quite a few locations, edges are found where there are no true object edges. This is the result of an incorrect block clustering, because if there is no true object edge in a pair of edge blocks, the algorithm will find an incorrect edge instead of the true edge. As a result parts of the clustering suffer from undersegmentation.

To compensate for this, we first have to try to detect this phenomenon. Because of the ECD, we know that between border pixels from different clusters there should be a discontinuity in colour. After clustering a pair of blocks we have information about the mean colour in these blocks for both clusters. From this we can calculate the distance between the mean colours of the present clusters. We will call this distance the *edge strength*. By specifying a threshold which the edge strength has to exceed, we can detect if there is a discontinuity in colour. In figure 6.4 the strong edges are displayed as white edges, and weaker edges are darker. You can see that in most locations where edges were incorrectly found, the edge strength is very low.

As we have now detected the edges along which the problems occur, we have to change the clustering here. Because the block clustering is incorrect in these blocks, we would ideally want to reassign one of the blocks to the correct cluster. Unfortunately these errors occur particularly often in homegenous areas. In these homogenous areas it is difficult if not impossible to say to which cluster the blocks belong. However, it is possible to indicate that this is an area which we

Figure 6.5: Frame 10 of the renata sequence. The left picture is the pixel clustering. The right picture is the same clustering after introducing new clusters in 25 iterations.Lines which are darker indicate that the difference between the means during pixel resolution clustering was low. Note how lines are only darker where the pixel resolution segmentation found an incorrect edge.

are unsure about. This would mean the application of our algorithm can determine what to do with such areas.

So, basically, we want to introduce a new cluster which represents an area about which we can not say anything. This is easily done by introducing a new cluster for each pair of blocks for which the edge strength is beneath the threshold. All pixels in the pair should be assigned to this block. If at any time the edge strength between two such clusters will be beneath the threshold these two clusters should be merged. Because this changes the clustering at a block resolution, it creates new border blocks as well. This means that we have to iterate over the screen again. From a performance perspective though, it is wise to restrict the maximum number of iteration. Fortunately, from testing it seems that after a limited number of iterations, no undersegmentation exists near the original weak edge anymore. See figure 6.5 for results. The next table lists results from the segmentation measure.

		Intra-block clustering	with introduction of new clusters
mummy	ϵ_o	0.60	0.66
	ϵ_u	0.26	0.21
	ϵ_s	0.86	0.86
football	ϵ_o	0.63	0.68
	ϵ_u	0.53	0.38
	ϵ_s	1.16	1.05
Total	ϵ_o	1.23	1.34
	ϵ_u	0.78	0.59
	ϵ_s	2.02	1.91

As you can see, oversegmentation increases which is inherent for the technique, but also undersegmentation decreases significantly. As oversegmentation is usually more desirable than undersegmentation, this seems to be a useful addition to the algorithm.

6.4 Smoothing the edges

As is visible in figure 6.5, the algorithm has difficulty finding edges if an object is in front of a detailed object. This is mostly because the colour cue is not sufficient in all pairs of blocks.

This led us to believe that using the morphology cue more explicitly might help. When observing objects with a certain minimum size, it came to our attention that objects display some amount of coherence.

To use this observation, we wanted to adjust our pixel resolution segmentation method, so that it would not only decrease the distance to the mean colour of a cluster for each pixel, but would also increase the coherence for a pixel. For this we had to define a measure for each pixel how coherent it is with other pixels from its cluster. As coherence indicates more of the neighbour pixels are of the same cluster we came up with the following measure. The coherence at a pixel (x, y) for cluster k are the number of pixels in a $n \times m$ neighbourhood around (x, y) which have label k. The parameters n and m are generically set for the complete algorithm. The values for $n \times m$ we have been experimenting with are 3×3, 5×5 and 7×7. Put in formal terms the coherence $c_{n\times m}(x, y)$ at location (x, y) is captured in the following equation. Because we want the neighbourhood to be symmetrical we only introduce it for odd values of n and m.

$$c_{n\times m}(p, k) = |\{q \mid (|p_{\mathrm{x}} - q_{\mathrm{x}}| < \frac{1}{2}n) \wedge (|p_{\mathrm{y}} - q_{\mathrm{y}}| < \frac{1}{2}m) \wedge (p_{\mathrm{n}} = q_{\mathrm{n}}) \wedge (S(q) = k)\}| \tag{6.3}$$

Modified to include this measure the algorithm becomes:

Algorithm K-shapes:
do $(\exists k : 0 \leq k <\mathrm{K}: \mu_k^{i-1} \neq \mu_k^i) \wedge (i < i_{\max}) \rightarrow$
foreach $0 \leq k <\mathrm{K}$ **do**
$S_k := \{p \mid (S(p) = k) \wedge (p \in D^n)\}$;
$\mu_k^i := 1/|S_k| \sum_{p\in S_k} C_{\mathrm{YUV}}(p)$;
od;
foreach $p \in \{q \mid (q_{\mathrm{b}} = b) \vee (q_{\mathrm{b}} = c)\}$ **do**
$C := \{k \mid (S(c) = k) \wedge c \in N_4(p)\}$;
$S(p) := \arg\min_{k\in C} m_{\mathrm{YUV}}(\mu_k^i, C_{\mathrm{YUV}}(p)) + rc_{n\times m}(p, k)$;
od;
$i := i + 1$;
od

Results are visible in figure 6.6. As you can see, the best choice for $n\times m$ is 3×3. From visual results we conclude that this is the best choice for any regularization parameter r. Especially together with the technique for adding new clusters introduced in the previous section this coherence measure achieves good results. These results are displayed in figure 6.7. The next table lists the results of the measure with various sizes of the smoothing area.

		Intra-block clustering	3×3 regularisation	5×5	7×7
mummy	ϵ_{o}	0.60	0.52	0.51	0.50
	ϵ_{u}	0.26	0.34	0.36	0.51
	ϵ_{s}	0.86	0.85	0.87	0.87
football	ϵ_{o}	0.63	0.57	0.57	0.58
	ϵ_{u}	0.53	0.60	0.63	0.64
	ϵ_{s}	1.16	1.17	1.20	1.22
Total	ϵ_{o}	1.23	1.09	1.08	1.08
	ϵ_{u}	0.79	0.94	0.99	1.15
	ϵ_{s}	2.02	2.02	2.07	2.09

As you can see, the *football* sequence actually suffers from using the coherence measure. We think this is the case because the objects in the sequence are too small. This could perhaps be remedied

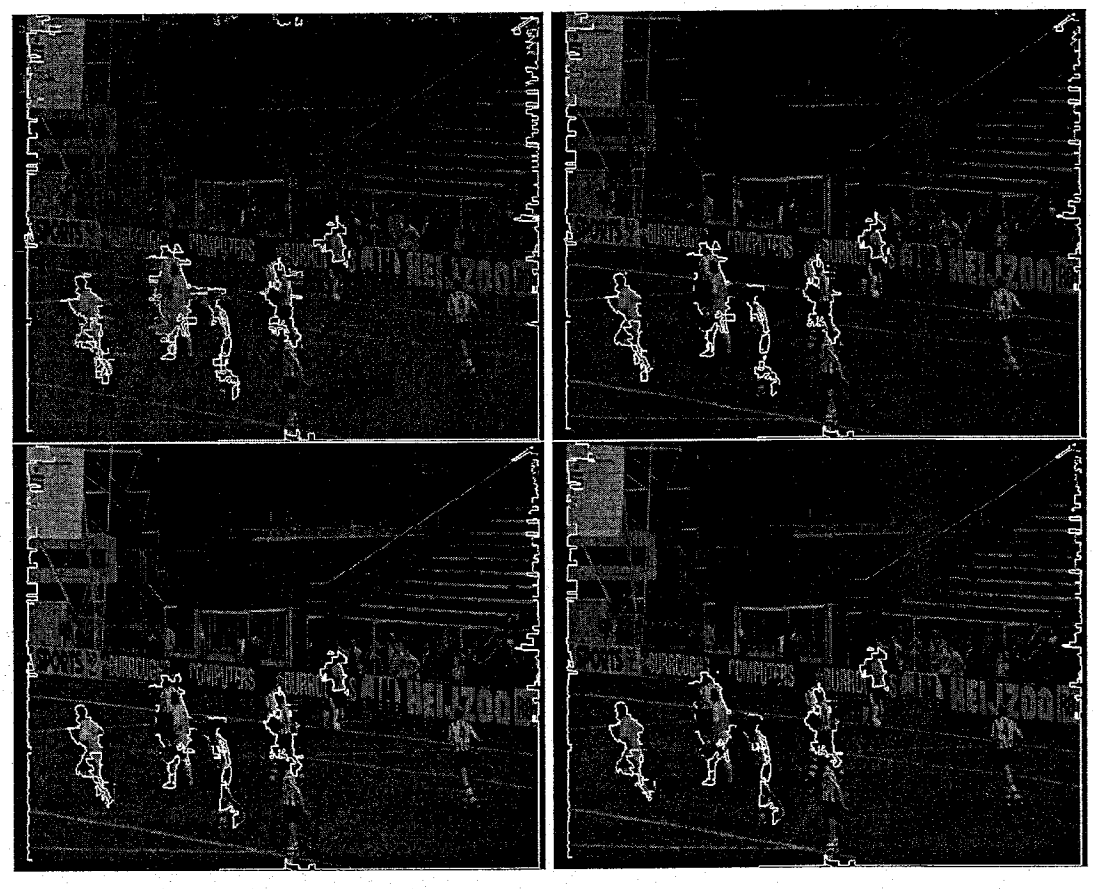

Figure 6.6: Frame 13 of the football sequence. Pixel resolution segmentation with the coherence measure and $r = 50$ without introducing new clusters. Top right picture is with regularisation in a 3×3 area, bottom left 5×5 and bottom right 7×7. Top left picture is pixel resolution segmentation without a coherence measure.

by somehow basing the regularization parameter r on the size of the objects involved. The next table lists the results with regularisation and the introduction of new clusters.

		Intra-block clustering	with regularisation	introducing new clusters	both combined
mummy	ϵ_o	0.60	0.52	0.66	0.63
	ϵ_u	0.26	0.34	0.21	0.40
	ϵ_s	0.86	0.85	0.86	1.04
football	ϵ_o	0.63	0.57	0.68	0.60
	ϵ_u	0.53	0.60	0.38	0.45
	ϵ_s	1.16	1.17	1.05	1.05
Total	ϵ_o	1.23	1.09	1.34	1.23
	ϵ_u	0.79	0.94	0.59	0.85
	ϵ_s	2.02	2.02	1.91	2.09

The bad results for the *mummy* sequence can be explained by the fact that it contains lots of similar colours and that perhaps the threshold for the introduction of new clusters is set too high for this sequence. This might indicate that the threshold should be based on the particular colours which are present in an image.

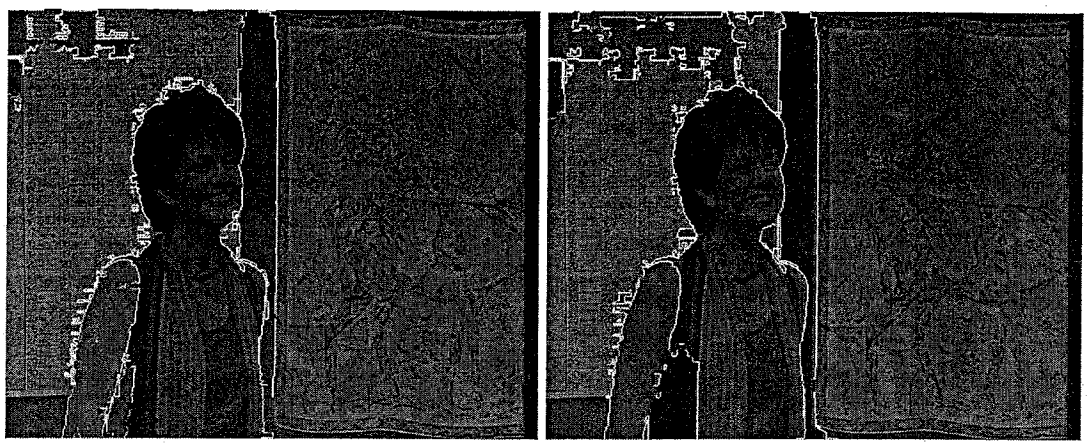

Figure 6.7: Frame 10 of the renata sequence. Pixel resolution segmentation with the introduction of new clusters. Left picture without coherence measure. Right picture with coherence measure and $r = 50$.

6.5 Reclassifying newly introduced clusters

Ideally, we would be able to find a measure which allows the clusters we introduced in areas with low edge strength to be reclassified to the right cluster. Because we already concluded that colour and motion field measures are insufficient, we could try to recreate the motion field for each new cluster. The best way to do is to try every motion model of every cluster the new cluster has replaced and see which matches the previous frame best when it is motion compensated. This is essentially the same as 3DRS-like motionestimators do but for a larger area instead for blocks. To use this method it is necessary to mantain a bookkeeping of which original clusters are (partly) replaced by the newly introduced clusters. We assume these original clusters to be contained in the set R_k for each cluster k. This adds a little more complexity to the introduction of new clusters. The reclassification algorithm looks like the following. The variables $a_{i,j}$ depict an array sized $K \times K$ with all values initialised to 0.

Algorithm Reclassify:
foreach $p \in D^n$**do**
foreach $k \in R_S(p)$**do**
$p' := p + \vec{d}^k_{\mathrm{p}}$;
$a_{S(p),k} := a_{S(p),k} + m(C(p), C(p'))$;
od;
od;
foreach $p \in D^n$**do**
$S(p) := \arg \min_{k \in R_S(p)} a_{S(p),k}$;
od;

Initial experiments with this technique indidicate that it is very fickle in its results. Sometimes, the results are exactly as you would hope them to be, at other times the results are exactly the opposite. Maybe the technique can be improved by specifying a minimum difference there should be between the first and second best distance. If the difference does not exceed the minimum, the newly introduced clusters should not be replaced. In figure 6.8 are some results.

Figure 6.8: Frame 10 of the renata sequence. Pixel resolution segmentation with coherence measure and the introduction of new clusters. Left picture without reclassification of newly introduced clusters. Right picture with reclassification.

6.6 Conclusion

The first, obvious, conclusion is that as this is a method which refines a block clustering, it is very important that this block clustering is correct. On the other hand, however, if the block clustering is incorrect the intra-block clustering will not further worsen the clustering significantly. So, all in all, intra-block clustering as a post processing step improves the results of block clustering. Also, the added complexity of doing intra-block clustering is not too high.

Also, we have found a way to partly detect locations of incorrect block clustering. Introducing new clusters at these locations improves the problems associated with undersegmentation, but inherently worsens oversegmentation. Our method of reclassifying these newly introduced clusters does not perform well enough for most applications. Indeed, it can be argued that our method of reclassification will regularly fail, because it is based on the same motion estimation principles which produced the incorrect block clustering in the first place. Additional research into such a reclassification method would be most welcome.

Of further note is that by constraining the colour segmentation with a coherence measure, the results improve. Looking for further ways to use the shape of objects to constrain the segmentation may be worthwhile. From testing, however, it became evident that the quality of the segmentation of smaller objects suffers from the coherence measure. This would imply that it might be a good idea to base the coherence measure also on the size of an object.

Another problem manifests itself when an object overlaps a highly detailled object. Often inaccurate edges will be found in those cases. This is especially true when certain parts of both objects have similar colours. This is probably caused by the fact that one pixel is not always enough context to determine which label should be assigned to the pixel. Perhaps this can be solved by adding extra resolution steps between block and pixel resolution.

Our pair-wise segmentation does not cover all configurations of edges, but nonetheless the quality of segmentation does not suffer as much as you would expect. However, if the pair-wise segmentation would be based on 8-connectedness, all different combinations of edges would be covered. Experiments with this might further increase the quality of the segmentation at the expense of more complexity.

Chapter 7

Conclusions

In this chapter we will first summarise the conclusions of the separate block and intra-block clustering steps in sections 7.1 and 7.2. In the following section, 7.3 we present our conclusions for the combination of both steps. Finally, in section 7.4 we present our conclusion for the measure for segmentation quality.

7.1 Block clustering

The major conclusion of using motion as a cue for block clustering is that motion indeed is a good cue for segmenting objects. It hardly suffers from the problems the colour cue has with oversegmentation, while also keeping undersegmentation to a minimum. Apparent motion is regularly less time stable than other cues though. While the colour of an object will only change marginally, the motion of an object typically changes quite much over small time periods. However, by doing motion compensation, it is feasible to track objects over time, which results in a better time stability than by using colour alone. This, however, is highly dependent on the accuracy of the motion estimator used.

Also, by further constraining clustering morphologically, better results are obtained. One of the advantages is that morphological constraints allowed us to use more complicated models for the motion of objects. This conclusion might be a worthwhile conlusion not only for motion segmentation but also for other segmentation algorithms where morphological constraints may be valid.

Despite these positive findings there are some problems we have encountered:

1. If two (or more) objects display the same apparent motion, there is no way to avoid undersegmentation. Note that if at some time, they display different motion, some temporal criterion may help to distinguish the objects. However, it may be very hard to find an appropriate criterion. This is an inherent problem of using motion as a cue.

2. The state of the art of motion estimation is not completely adequate for segmentation. Errors in the vector field cause problems with clusters containing lots of detail stealing blocks from other clusters with less detail. These errors also cause problems with new clusters appearing, while there is no reason for them to appear.

3. Furthermore, current motion estimation methods only return a vector per block, while a vector per pixel would be necessary for a pixel resolution video segmentation. Note that the methods we presented in chapter 5 will work with blocks of any size, including blocks sized

one pixel. However, the current motion estimators force us to do a clustering on a block resolution.

4. Also, the accuracy of the motion vectors is not high enough, to reliably detect or distinguish small motion. This is especially a problem with objects moving towards the camera, because their motion will seem to differ in a relatively small way from the surrounding motion.

5. The block clustering method suffers from low temporal stability when objects reappear after total occlusion. This problem exists because information is stored only from one previous frame. Solving this problem would require the algorithm to store information on occluded objects and a technique to recognise objects from this information. The alternative we use now is to consider these objects as being new previously unseen objects.

6. The clusters representing objects with relatively small extremities, such as people at a distance, will often not encompass these extremities. This is probably because these extremities only encompass a relatively small part of a block, prohibiting 3DRS of finding the right vector for that block.

7. Merging and splitting produces, depending on the thresholds, under- or oversegmentation. As stated before this is most probably caused by the inexact definition of what an object is.

8. Large lowly detailled areas tend to be oversegmented, because the motion estimator returns incorrect vectors for these areas. This is caused because the motion estimators are based on matching and blocks with low detail do not offer enough information to find a reliable match in the previous frame.

7.2 Intra-block clustering

From chapter 6 we can conclude several things. The first, obvious, conclusion is that as this is a method which refines a block clustering, it is very important that this block clustering is correct. On the other hand, however, if the block clustering is incorrect the intra-block clustering will not further worsen the clustering significantly. So, all in all, intra-block clustering as a post processing step improves the results of block clustering. Also, the added complexity of doing intra-block clustering is not too high.

Also, we have found a way to partly detect locations of incorrect block clustering. Introducing new clusters at these locations improves the problems associated with undersegmentation, but inherently worsens oversegmentation. Our method of reclassifying these newly introduced clusters does not perform well enough for most applications. Indeed, it can be argued that our method of reclassification will regularly fail, because it is based on the same motion estimation principles which produced the incorrect block clustering in the first place. Additional research into such a reclassification method would be most welcome.

Of further note is that by constraining the colour segmentation with a coherence measure, the results improve. Looking for further ways to use the shape of objects to constrain the segmentation may be worthwhile. From testing, however, it became evident that the quality of the segmentation of smaller objects suffers from the coherence measure. This would imply that it might be a good idea to base the coherence measure also on the size of an object.

Another problem manifests itself when an object overlaps a highly detailled object. Often inaccurate edges will be found in those cases. This is especially true when certain parts of both objects have similar colours. This is probably caused by the fact that one pixel is not always enough context to determine which label should be assigned to the pixel. Perhaps this can be solved by adding extra resolution steps between block and pixel resolution.

Our pair-wise segmentation does not cover all configurations of edges, but nonetheless the quality of segmentation does not suffer as much as you would expect. However, if the pair-wise segmentation would be based on 8-connectedness, all different combinations of edges would be covered. Experiments with this might further increase the quality of the segmentation at the expense of more complexity.

7.3 Block and intra-block clustering combined

Our two step approach for segmentation does combine some of the better features of both motion and colour segmentation. Especially the tracking of objects is, especially due to the use of motion compensation, a lot better than usual colour segmentation techniques. Similarly, colour segmentation enhances the motion segmentation by allowing us to find edges at pixel resolution. Also, the local approach of the colour segmentation, which is enabled by the motion based block clustering enhances the quality of the colour segmentation, because colours tend to vary little locally. The colour segmentation allows us to detect blocks in which the block clustering has failed. Our two step approach allows us to do a pixel resolution segmentation with relatively little added complexity over the block resolution segmentation by using the strong points of both types of segmentation.

However, there are some disadvantages apart from those listed in the previous conclusions. Because our first step is motion segmentation, objects which do not move are never assigned to their own clusters. Similarly, two objects which display the same movement get assigned to the same clusters even though their colours may be very different. These problems may not be real problems for some of the applications though.

As a combination, the results of the algorithm are especially dependent of the first block clustering step. With the introduction of new clusters from section 6.3, though, we have managed to weaken this dependency somewhat. This introduction does produce more oversegmentation though, but not nearly as much as conventional colour segmentation techniques. Although this helps somewhat, any faults in the first block clustering step are still the real weak point of the algorithm. A second weak point is the trouble the pixel resolution has with high amounts of detail in the background.

Also, during our research a variety of applications have been thought of our algorithm, even though we have not designed it for one particular applicatioin. This is, to our opinion, demonstrative of how desirable a good segmentation algorithm is. Adapting our algorithm to more specific applications may be an important next step.

7.4 Measuring segmentation quality

The segmentation quality measure seems to be inconclusive because of the limited test set that we have created. Although the concept still seems promising, results from the quality measure can only be considered significant when a larger set of ground truth frames is available.

Chapter 8

Suggestions for further study

In this chapter we will discuss new ideas and improvements for the algorithm which warrant new research. First, we will formulate some general ideas for improving research into video segmentation. In the second section we will comment on new ideas for block clustering. In the third section the ideas for the pixel resolution clustering will be covered and in the fourth section we will suggest further study for the measurement of segmentation quality.

8.1 General ideas

In general, research should be done in further making the problem more specific. Especially the notion of what an object is, is unclear at the moment. For this, it will probably be necessary to develop video segmentation techniques for a more specific application than ours.Nonetheless, it will not be easy to find such a specification.

Several concepts in this report, like K-Models and our two step approach, are usable not only for video segmentation, but for other types of data as well. Adapting this into a generic segmentation method for similar data may be feasible.

8.2 Improvements for our block resolution clustering techniques

There are several ideas that deserve further attention. Firstly, a feed back into the motion estimator would allow us to adjust and smooth the results of the motion estimator and might recursively prevent some errors resulting from incorrect block clustering. In our opinion this step is one of the most promising steps forward for the improvement of our block clustering technique. Also, the initialization of the block clustering is performed with the results of the previous block clustering as input. Using the results of the intra-block clustering instead, might also help to recursively prevent some errors.

The motion models could be extended to include a dependency on time as well. This could increase time stability. Other ideas to increase time stability include the use of information from the colour or morphology cues as an identifying feature for clusters. Shapes and colours of most objects vary little over time.In the pixel resolution clustering we noticed that using an extra coherence measure improved the correctness of the clustering. It seems like a good idea to try to if this works equally well at block resolution.

During some experiments in which we studied the effectiveness of this algorithm as a motion estimation algorithm, we found that the least squares method for fitting the models may not be ideal, because the results are affected too much by outliers. Also, as noted in section 3.1, the ratio for the scaling parameters is fixed. A model which assumes that this ratio is fixed for all of the clusters' motion models in a sequence would be more constrained and thus suffer from less undersegmentation.

8.3 Improvements for our pixel resolution clustering techniques

The distinguishing of two objects next to eachother containing lots of detail is most difficult problem found so far. Perhaps by starting our colour segmentation not at pixel resolution, but at a resolution of blocks which are sized 1/4 of a normal block, we can use a bigger context for colour segmentation and thus ignore detail. Each of these *mini-blocks* could be clustered based on the distance of its mean to the cluster's mean.

Using a coherence measure improved the correctness a lot, but we experienced undersegmentation problems with smaller objects. Perhaps using the coherence measure to a lesser degree with smaller objects would fix these problems and would allow us to use more morphology cues for pixel resolution segmentation. Using the general shape of an object in some measure could be a good idea. After all, shape information about the block clustering is already available.

The reclassification of newly introduced clusters is as yet too unpredictable. Research should be done in making this technique more robust. Using additional cues for this might help.

8.4 Ideas for the measurement of segmentation quality

This is an area in which few research has been done, while it deserves much more attention. In our opinion a good measure of segmentation quality would help research into video segmentation significantly. We think it might be easier to devise a good measure if the segmentation is intended for one specific application. In that case, the segmentation itself does not need to be measured, but the system can be evaluated as a whole. This means it should be evaluated to what degree the system matches its specifications. This is often easier to quantify than the specification for segmentation. Also, to better use our segmentation quality measure, a bigger ground truth test set is needed. Extension of our current test set is also worthy of extra effort. Perhaps tools should be created to ease the creation of such a test set.

Bibliography

[1] P.E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

[2] G. de Haan. *Video Processing for multimedia systems*, pages 192–193. University Press Eindhoven, 2000.

[3] G. de Haan and P.W.A.C. Biezen. Sub-pixel motion estimation with 3-D recursive search block-matching. *Signal Processing: Image Communication*, 6:229–239, 1994.

[4] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[5] G.A. Lunter. Block-based motion estimation: Some methods to deal with occlusion and convergence. Technical Report TN2002/076, Philips Research Laboratories, February 2002.

[6] J. McQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Sympos. on Math. Stat. and Prob.*, volume 1, pages 773–785, May 1979.

[7] F. Ernst P. Willinski and K. van Overveld. Structure from motion: depth estimation based on matching. Technical Report TN2001/343, Philips Research Laboratories, August 2001.

[8] C. Varekamp. Temporal stability of surface edge positions derived from video. *Advanced Concepts for Intelligent Vision Systems*, Yet to be published.